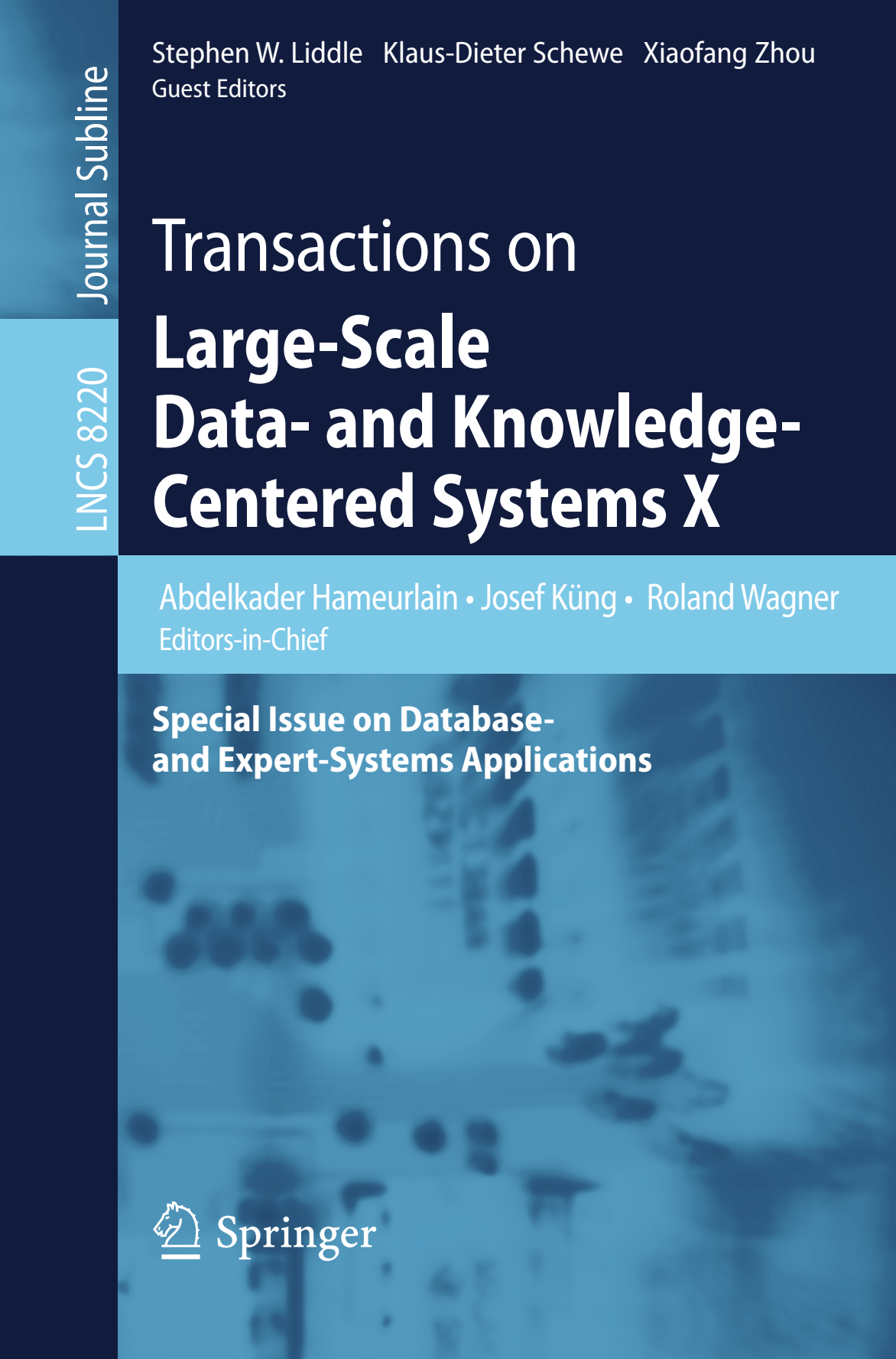
Journal Subline
LNCS 8220

Stephen W. Liddle Klaus-Dieter Schewe Xiaofang Zhou
Guest Editors

Transactions on Large-Scale Data- and Knowledge-Centered Systems X

Abdelkader Hameurlain • Josef Küng • Roland Wagner
Editors-in-Chief

Special Issue on Database- and Expert-Systems Applications



Springer

Lecture Notes in Computer Science 8220

Commenced Publication in 1973
Founding and Former Series Editors:
Gerhard Goos, Juris Hartmanis, and Jan van Leeuwen

Abdelkader Hameurlain Josef Küng
Roland Wagner Stephen W. Liddle
Klaus-Dieter Schewe Xiaofang Zhou (Eds.)

Transactions on Large-Scale Data- and Knowledge-Centered Systems X

Special Issue on Database- and Expert-Systems Applications

Springer

ISSN 0302-9743 (LNCS) e-ISSN 1611-3349 (LNCS)
ISSN 1869-1994 (TLDKS)
ISBN 978-3-642-41220-2 e-ISBN 978-3-642-41221-9
DOI 10.1007/978-3-642-41221-9
Springer Heidelberg New York Dordrecht London

Library of Congress Control Number: 2013948623

CR Subject Classification (1998): H.2.8, H.2, I.2.6, H.3, D.2, C.2, F.2, J.1

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India

Printed on acid-free paper

Springer is part of Springer Science+Business Media (www.springer.com)

Preface

Following the 23rd International Conference on Database and Expert Systems Applications (DEXA 2012, proceedings published in LNCS volumes 7446 and 7447) the programme chairs invited the authors of twelve selected papers to submit original, extended and revised papers to a special issue of the Transactions on Large-Scale Data- and Knowledge-Centered Systems (TLDKS). Following these invitations ten papers were submitted, each of which was carefully reviewed by two experts in the field in two reviewing rounds. Finally, seven papers were accepted for this special issue.

In their paper "Stepwise Development of Formal Models for Web Services Compositions" Idir Ait-Sadoune and Yamine Ait-Ameur deal with modelling and property verification for composed web services. In general, the composition process for web services is defined by a choreography and orchestration of atomic services. These compositions are seen as state transition systems describing the communication protocol between the participating services. The description languages for service compositions suffer from a lack of formal semantics and from ambiguities in the definition of their constructors, and the associated tools neglect formal verification and validation of the behaviour and the properties of the composed services. The article focuses on the modelling and verification of composed web services described by BPEL standard using the *Event-B* method. The static and dynamic parts of BPEL are formalised by Event-B, and formal refinement is used to structure the development with BPEL. A one-to-one link is guaranteed between BPEL elements and their Event-B formalisation. This correspondence provides assistance for developers to improve the quality of the obtained BPEL process. Properties are verified by proving theorems, and the whole approach is implemented in the BPEL2B tool.

The article "Computing Skyline Incrementally in Response to Online Preference Modification" by Tassadit Bouadi, Marie-Odile Cordier and René Quiniou is dedicated to skyline queries, which are understood as queries retrieving the most interesting objects from a database with respect to multi-dimensional preferences. The presented EC2Sky approach focuses on how to answer efficiently skyline queries in the presence of dynamic user preferences and large volumes of data. The approach exploits the fact that the skyline associated with any preference on a particular dimension can be computed without domination tests from the skyline points associated with first-order preferences on that particular dimension. This can be supported by IPO-Trees (Implicit Preference Order Trees), a data structure that supports the materialisation of skyline points associated with the most preferred values. However, the size of an IPO-tree grows exponentially with the number of dimensions. Therefore, the authors developed an incremental method for calculating the skyline points related to several dimensions associated with dynamic preferences. For this purpose a materialisation of

linear size is sufficient, which permits greater flexibility for updates of dimension preferences and improves the execution time and storage size of queries. Experiments on synthetic data highlight the relevance of EC2Sky compared with the IPO-Tree method.

The third article in this collection by Flavio Ferrarotti, Sven Hartmann, Sebastian Link, Mauricio Marin and Emir Muñoz handles foundations, applications and performance issues of 'The finite Implication Problem for Expressive XML Keys". The theoretical contribution is the definition of a new fragment of XML keys that keeps the right balance between expressiveness and efficiency of maintenance. More precisely, they characterise the associated implication problem axiomatically and develop a low-degree polynomial time decision algorithm. In comparison to previous work this new fragment of XML keys enhances the possibility of capturing properties of XML data that are significant for an application at hand. The practical contribution includes an efficient implementation of this decision algorithm and a thorough evaluation of its performance, demonstrating that reasoning about expressive notions of XML keys can be done efficiently in practice, and scales well. The results promote the use of XML keys on real-world XML practice, where a little more semantics makes applications a lot more effective. To exemplify this potential, the decision algorithm is used to calculate non-redundant covers for sets of XML keys. In turn, this permits a significant reduction of the the time required to validate large XML documents against keys from the proposed fragment.

In their article "ALACRITY: Analytics-Driven Lossless Data Compression for Rapid In-Situ Indexing, Storing, and Querying" John Jenkins and his coauthors present ALACRITY, an approach to effectiveness of a fused data and index encoding of scientific, floating-point data in generating lightweight data structures that are amenable to common types of queries used in scientific data analysis. The authors exploit the representation of floating-point values by extracting significant bytes, using the resulting unique values to bin the remaining data along fixed-precision boundaries. To optimise query processing an inverted index is used mapping each generated bin to a list of records contained within, which optimises query processing with attribute range constraints. Overall, the storage footprint for both index and data is shown to be below numerous configurations of bitmap indexing while matching or outperforming query performance.

The article "A Declarative Approach to View Selection Modeling" by Imene Mami, Zohra Bellahsene and Remi Coletta deals with the important view selection problem in database and data warehousing systems. Given a database (or a data warehouse) schema and a query workload, the view selection problem is to choose an appropriate set of views to be materialised such that the total query costs are optimised under constraints such as limited amount of resources and total view maintenance costs. The view selection problem is known to be NP-complete. The new contribution of the authors is a declarative approach that involves a constraint programming technique known for its efficiency for

the resolution of NP-complete problems. The view selection problem is modeled as a constraint satisfaction problem in an easy and declarative way, and its resolution is performed automatically by the constraint solver. The approach guarantees more flexibility and is extensible, as it can model and handle new constraints and new heuristic search strategies to reduce the solution space. The authors show that the performance outperforms genetic algorithms, which are known to provide the best trade-off between quality of solutions in terms of cost saving and execution time.

In the article "A Framework for Modeling, Computing and Presenting Time-Aware Recommendations" Kostas Stefanidis and his co-authors deal with recommendation systems. While many existing approaches recommend items of potential interest to users by completely ignoring the temporal aspects of ratings, the authors argue that time-aware recommendations need to be pushed to the foreground. Therefore, they introduce an extensive model for time-aware recommendations from two perspectives. From a fresh-based perspective, the use of different aging schemes for decreasing the effect of historical ratings and increasing the influence of fresh and novel ratings is proposed. From a context-based perspective, the focus is on the provision of different suggestions under different temporal specifications. In addition, to facilitate user browsing, an effective presentation layer for time-aware recommendations based on user preferences and summaries for the suggested items has been developed.

Finally, the last article in this collection is "Incremental Mining of Top-k Maximal Influential Paths in Network Data" by Enliang Xu, Wynne Hsu, Mong Li Lee and Dhaval Patel dealing with information diffusion, which refers to the spread of abstract ideas and concepts, technical information, and actual practices within a social system. The spread denotes flow or movement from a source to an adopter, typically via communication and influence. While a lot of research on information diffusion analysis has focused on discovering "influential users" and "who influences whom" neglecting the continuity of influence among users, the authors develop a new method for inferring top-k maximal influential paths capturing the continuity of influence. For this a generative influence propagation model is defined based on the independent cascade model and the linear threshold model, which mathematically models the spread of certain information through a network. The top-k maximal influential path inference problem is formalised giving rise to an efficient algorithm called TIP to infer the top-k maximal influential paths. TIP makes use of the properties of top-k maximal influential paths to dynamically increase the support and prune the projected databases. In order to address database evolution over time an incremental mining algorithm IncTIP has been developed in addition to maintain top-k maximal influential paths. Effectiveness and efficiency of both TIP and IncTIP are evaluated by a case study on both synthetic and real-world datasets.

We would like to thank all authors for their contributions to this special issue. We are grateful to all reviewers for their invaluable work in reviewing the papers and ensuring the high quality of this collection of articles. Finally, without the editorial assistance by Gabriela Wagner, who handled all the communication with the authors and the reviewers, this volume would not have been possible.

July 2013

Stephen W. Liddle
Klaus-Dieter Schewe
Xiaofang Zhou

Editorial Board

Reviewers

Table of Contents

Stepwise Development of Formal Models for Web Services Compositions: Modelling and Property Verification

Idir Ait-Sadoune[1] and Yamine Ait-Ameur[2]

[1] E3S - Supelec, Gif-Sur-Yvette, France
idir.aitsadoune@supelec.fr
[2] IRIT - ENSEEIHT, Toulouse, France
yamine@n7.fr

Abstract. The ability to compose existing services to provide more complex features is one of the main benefits of Service Oriented Architecture (SOA). This services composition process, especially Web services, is generally defined by a choreography or an orchestration of atomic services. These compositions are seen as a state transition system describing the communication protocol between the participating services. The services description languages, expressing these compositions, suffer from the lack of formal semantics and the ambiguities in the definition of their constructors in the standards defining these languages. The tools associated with these languages do not offer the possibility to formally verify and validate the behaviour and the properties of the obtained composed service.

Our work focuses on the formal modelling and verification of the web services composition described by the BPEL standard using the Event-B method. The proposed approach formalizes the static and the dynamic parts of BPEL, and uses the refinement to structure a BPEL development. The theorem-proving technique is set-up to verify the properties. A one-to-one link is guaranteed between BPEL elements and their Event-B formalization. This correspondence provides assistance for developers to improve the quality of the obtained BPEL process. This approach is implemented in the BPEL2B tool.

Keywords: Formal modelling, Verification, Refinement and proof, Event-B, Composition operators, Services composition.

1 Introduction

With the development of the web, a huge number of services available on the web have been published. These web services operate in several application domains like concurrent engineering, semantic web, system engineering or electronic commerce. Moreover, due to the ease of use of the web, the idea of composing these web services to build composite ones defining complex workflows arose. Even if several industrial standards providing specification and/or design XML-oriented

A. Hameurlain et al. (Eds.): TLDKS X, LNCS 8220, pp. 1–33, 2013.

languages for web services compositions description, like BPEL [27], CDL [35], OWL-S [33], BPMN [28] or XPDL [36] have been proposed, the activity of composing web services remains a syntactically based approach. Due to the lack of formal semantics of these languages, ambiguous interpretations remain possible and the verification of the compositions is left to the testing and deployment phases. From the business point of view, customers do not trust these services nor rely on them. As a consequence, building correct, safe and trustable web services compositions is a major challenge.

This paper presents an extended version of the work addressed in [8, 10]. It gives a detailed, integrated and complete description of the approach consisting in formally interpreting web services compositions described with BPEL. Moreover, it covers new concepts related to the methodology advocated by our proposal. From the *methodological point of view*, the refinement based approach defined in [10] is extended by defining four good candidates scenarios that formalize the BPEL decomposition operator. As a consequence, a stepwise methodology based on decomposition is described in this paper. Our objective is not to change the design process mastered by the web services compositions designers, but it is to enrich this process with a formal modelling and verification activity by providing formal models beside each web services composition description. More precisely, our approach consists in deriving an Event-B model from each BPEL web service compositions and encode the decomposition relationship, offered by BPEL, by an Event-B refinement relationship. Therefore, it becomes possible to enrich the Event-B models by the relevant properties in order to check not only the correctness of the web services compositions described in the BPEL designs but also to check the correctness of the decomposition encoded by the sequence of BPEL models linked by the decomposition relationship. Regarding the formal modelling activity, the complexity of the proofs is reduced thanks to refinement.

From a *technical point of view*, we are aware that several work addressed the problem of formal web services verifications [11]. These approaches use model checking as a verification procedure facing the state number explosion problem. Abstraction techniques are used in order to reduce the complexity of the verification. The consequence of this abstraction is the loss of relevant properties modelling and verification like data interpretation properties or message exchanges. We claim that, thanks to the Event-B method, we are capable to overcome this limitation by supplying a technique supporting data interpretations. In this paper, we develop the idea that consists in providing a single and formal interpretation of WSDL/BPEL constructs by Event-B models, by giving different Event-B encodings that formalize WSDL/BPEL elements.

This paper is structured as follows. Section 2 presents the Event-B method. Section 3 is devoted to a brief description of web services composition and BPEL standard. Then, Section 4 describes our approach for formalizing any BPEL design by an Event-B model. In Section 5, we show how the refinement offered by Event-B can be used to encode the BPEL decomposition operation. As a result, we obtain an incremental development of both the BPEL design and the Event-B models. Section 6 discusses the properties verification capabilities that

result from this approach and Section 7 presents the BPEL2B tool that implements the proposed approach. In section 8, we discuss the existing approaches for formal verification of web services compositions and we compare them with our proposal. Finally, a conclusion and some perspectives are outlined.

2 The Event-B Method

The Event-B method [3] is a recent evolution of the B method [2]. This method is based on the notions of pre-conditions and post-conditions of Hoare [21], the weakest pre-condition of Dijkstra [14] and the substitution calculus [2]. It is a formal method based on mathematical foundations: first order logic and set theory.

2.1 Event-B Model

An Event-B model is defined by a set of variables, defined in the VARIABLES clause that evolve thanks to events defined in the EVENTS clause. It encodes a state transition system where the variables represent the state and the events represent the transitions from one state to another.

An Event-B model is made of several components of two kinds : Machines and Contexts. The Machines contain the dynamic parts (states and transitions) of a model whereas the Contexts contain the static parts (axiomatization and theories) of a model. A Machine can be refined by another one, and a Context can be extended by another one. Moreover, a Machine can see one or several Contexts (figure 1).

A Context is defined by a set of clauses (figure 2) as follows.

- CONTEXT represents the name of the component that should be unique in a model.
- EXTENDS declares the Context extended by the described Context.
- SETS describes a set of abstract and enumerated types.
- CONSTANTS represents the constants used by a model.

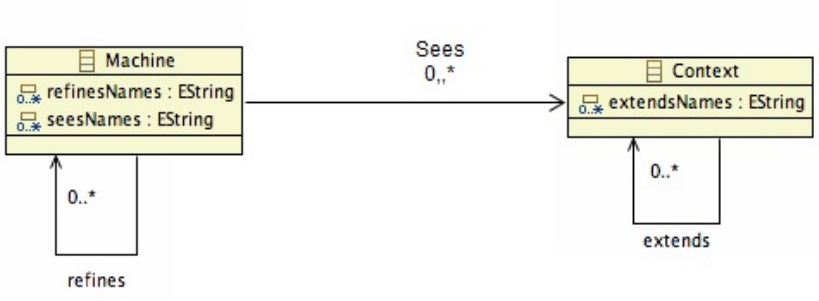


Fig. 1. MACHINE and CONTEXT relationships

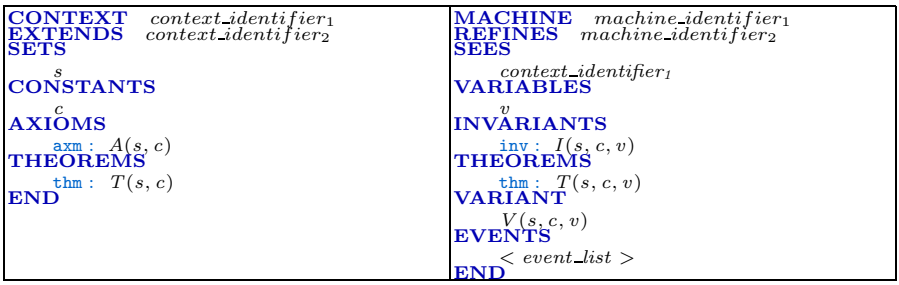


Fig. 2. The structure of an Event-B development

- AXIOMS describes, in first order logic expressions, the properties of the attributes defined in the CONSTANTS clause. Types and constraints are described in this clause as well.
- THEOREMS are logical expressions that can be deduced from the axioms.

Similarly to Contexts, a Machine is defined by a set of clauses (figure 2). Briefly, the clauses mean.

- MACHINE represents the name of the component that should be unique in a model.
- REFINES declares the Machine refined by the described Machine.
- SEES declares the list of Contexts imported by the described Machine.
- VARIABLES represents the state variables of the model of the specification. Refinement may introduce new variables in order to enrich the described system.
- INVARIANTS describes, by first order logic expressions, the properties of the variables defined in the VARIABLES clause. Typing information, functional and safety properties are usually described in this clause. These properties shall remain true in the whole model. Invariants need to be preserved by events. It also expresses the gluing invariant required by each refinement.
- THEOREMS defines a set of logical expressions that can be deduced from the invariants. They do not need to be proved for each event like for the invariant.
- VARIANT introduces a decreasing natural number which states that the "convergent" events terminate.
- EVENTS defines all the events (transitions) that occur in a given model. Each event is characterized by its guard and is described by a body thanks to actions. Each Machine must contain an "*Initialisation*" event. The events occurring in an Event-B model affect the state described in VARIABLES clause.

  An event consists of the following clauses (figure 3):
  - **status** can be "ordinary", "convergent" (the event has to decrease the variant), or "anticipated" (the event must not increase the variant).
  - **refines** declares the list of events refined by the described event.

- **any** lists the parameters of the event.
- **where** expresses the guard of the event. An event is fired when its guard evaluates to true. If several guards evaluate to true, only one is fired with a non deterministic choice.
- **then** contains the actions of the event that are used to modify variables.

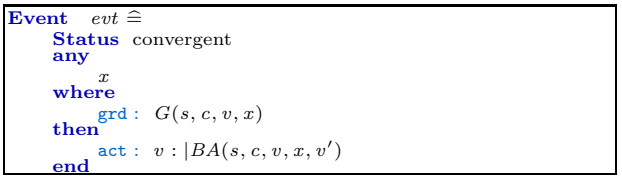


Fig. 3. Event structure

Event-B offers three kinds of actions that can be deterministic or not (figure 4). For the first case, the deterministic action is represented by the "*assignment*" operator that modifies the value of a variable. This operator is illustrated by the action [1]. For the case of the non-deterministic actions, the action [2] represents the "*before-after*" operator acting on a set of variables whose effect is represented by a predicate, expressing the relationship between the contents of variables before and after the triggering of the action. Finally, the action [3] represents the indeterministic choice operator, acting on a variable, by modifying its content with an undetermined value in a set of values.

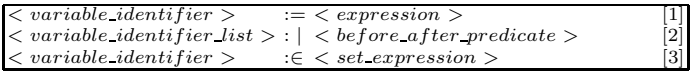


Fig. 4. The kinds of actions of an event

Proof obligations (PO) are associated to any Event-B model. They are automatically generated. *The proof obligation generator plugin* in the Rodin platform [29] is in charge of generating them. These OP need to be proved in order to ensure the correctness of developments and refinements. The rules for generating proof obligations are given in [3], they follow the substitutions calculus [2] close to the weakest precondition calculus [14].

2.2 Refinement of Event-B

The refinement operation [4] offered by Event-B encodes model decomposition. A transition system is decomposed into another transition system with more and more design decisions while moving from an abstract level to a less abstract one. A refined Machine is defined by adding new events, new state variables and a gluing invariant. Each event of the abstract model is refined in the concrete

model by adding new information expressing how the new set of variables and the new events evolve. All the new events appearing in the refinement refine the skip event of the refined Machine. Refinement preserves the proved properties and therefore it is not necessary to prove them again in the refined transition system, usually more complex. The preservation of the properties results from the proof of the gluing invariant in the refined machine.

2.3 Semantics of Event-B Models

The new aspect of the Event-B method, in comparison with classical B, is related to the semantics. Indeed, the events of a model are atomic events of a state transitions systems. The semantics of an Event-B model is trace based semantics with interleaving. A system is characterized by the set of licit traces corresponding to the fired events of the model which respects the described properties. The traces define a sequence of states that may be observed by properties. All the properties will be expressed on these traces.

This approach proved the capability to represent event based systems like railway systems, embedded systems or web services. Moreover, decomposition (thanks to refinement) supports building of complex systems gradually in an incremental manner by preserving the initial properties thanks to the preservation of a gluing invariant.

3 Services Composition Description Languages

One of the key ideas arising from the use of SOA architecture based on web services is the ability to create a new web service by combining, composing and interacting other pre-existing services. There exists important standardization efforts towards providing specification languages for web services composition. Our work deals with the formal verification of the composition of web services. We focus on verification of behavioural requirements, the properties that a service composition shall satisfy in order to achieve its functional goal. Currently, languages dedicated to the web services composition description like BPEL [27], CDL [35], OWL-S [33], BPMN [28] or XPDL [36] are not equipped with verification procedures that cover the whole languages. In order to illustrate our proposal, we use BPEL for describing web services compositions.

3.1 Services Composition Description Languages

The web services composition description languages are XML-based languages. The most popular languages are BPEL, CDL, OWL-S, BPMN or XPDL. If these languages are different from the description point of view, they share several concepts in particular the service composition. Among the shared concepts, we find the notions of *activity* for producing and consuming messages, *attributes* for instance correlation, message decomposition, service location, compensation

in case of failure, events and event handling. These elements are essential to describe services compositions and their behaviour.

However, due to their XML-based definition, these languages suffer from a lack of semantics. It is usually informally expressed in the standards that describe these languages using natural language or semi-formal notations. Hence the need of a formal semantics expressing this semantics emerged. Formal description techniques and the corresponding verification procedures are very good candidates for expressing the semantics and for verifying the relevant properties.

The formal approach we develop in this paper uses BPEL as a service composition description language and Event-B as a formal description technique. The proposed approach can be extended to be used with other service composition description language.

3.2 Overview of BPEL

BPEL (Business Process Execution Language [27]) is a standardized language for specifying the behaviour of a business process based on interactions between a process and its service partners (*partnerLink*). It defines how multiple service interactions, between these partners, are coordinated to achieve a given goal. Each service offered by a partner is described in a WSDL document through a set of *operations* and of handled *messages*.

WSDL (Web Service Description Language [34]) is a standardized language for describing the published interface (input and output parameters types) of the web service. It provides with the the address of the described service, its identity, the operations that can be invoked, the operation parameters and their types. Thus, WSDL describes the function provided by web service operations. It defines the exchanged messages the envelop the exchanged data and parameters. The composition of such web services is described in languages, like BPEL, supporting such composition operators.

A BPEL process uses a set of *variables* to represent the messages exchanged between partners. They also represent the state of the business process. The content of these messages is amended by a set of *activities* which represent the process flow. This flow specifies the operations to be performed, their ordering, activation conditions, reactive rules, etc. Figure 5 shows the XML structure of a WSDL description and a BPEL process.

BPEL offers two categories of activities: 1) atomic activities representing the primitive operations performed by the process. They are defined by the *invoke*, *receive*, *reply*, *assign*, *terminate*, *wait* and *empty* activities, they correspond to basic web services 2) and, structured activities obtained by composing primitive activities and/or other structured activities using the *sequence*, *if*, *while* and *repeat Until* composition operators that model traditional sequential control constructs. Three other composition operators are defined by the *pick* operator defining a non-deterministic choice, the *flow* operator defining the concurrent execution and the *scope* operator defining sub-processes execution. BPEL also introduces systematic mechanisms for fault handling by defining a set of activities to be executed for handling possible errors anticipated by the web services

```
<definitions name="PurchaseOrder" ...>

    <!--the messages declaration---------------------- -->
    <wsdl:message name="POMessage">
        <wsdl:part name="customerInfo" type="sns:customerInfoType" />
        <wsdl:part name="purchaseOrder" type="sns:purchaseOrderType" />
    </wsdl:message>

    <wsdl:message name="InvMessage">
        <wsdl:part name="IVC" type="sns:InvoiceType" />
    </wsdl:message>

    ...
    <!-- the operations description ---------------------- -->
    <wsdl:portType name="purchaseOrderPT">
        <wsdl:operation name="sendPurchaseOrder">
            <wsdl:input message="pos:POMessage" />
            <wsdl:output message="pos:InvMessage" />
        </wsdl:operation>
    </wsdl:portType>

</definitions>
```

```
<process name="purchaseOrderProcess" ... >
...
<!-- the state variables declaration ------------------------------------- -->
<variables>
    <variable name="PO" messageType="lns:POMessage"/>
    <variable name="Invoice" messageType="lns:InvMessage"/>
    <variable name="shippingRequest" messageType="lns:shippingRequestMessage"/>
    <variable name="shippingInfo" messageType="lns:shippingInfoMessage"/>
    <variable name="shippingSchedule" messageType="lns:scheduleMessage"/>
</variables>
<!-- the fault handler behavior description ------------------------------------ -->
<faultHandlers>
    <catch faultName="lns:cannotCompleteOrder"
           faultVariable="POFault" faultMessageType="lns:orderFaultType">
        <reply ... operation="sendPurchaseOrder" variable="POFault" />
    </catch>
</faultHandlers>
<!-- a main behavior of a purchase order process ---------------------------------- -->
<sequence name="PurchaseOrderProcess">
    <receive name="ReceiveOrder" ... operation="sendPurchaseOrder" variable="PO"/>
    <flow name="PurchaseOrderProcessing">
        <sequence name="ArrangeLogistics">
            <assign name="AssignCustomer_Info"...
                   <copy><from>$PO.customerInfo</from>
                   <to>$shippingRequest.customerInfo</to></copy>
            </assign>
            <invoke name="RequestShipping"... operation="requestShipping"
                    inputVariable="shippingRequest" outputVariable="shippingInfo"/>
            <receive name="ReceiveSchedule"...
                     operation="sendSchedule" variable="shippingSchedule"/>
        </sequence>
        <sequence name="ComputePrice">
            <invoke name="initiatePriceCalculation"...
                     operation="initiatePriceCalculation" inputVariable="PO"/>
            <invoke name="ReceiveShipping_Price"...
                     operation="sendShippingPrice" inputVariable="shippingInfo"/>
            <receive name="ReceiveInvoice"...
                     operation="sendInvoice" variable="Invoice" />
        </sequence>
        <sequence name="ProductionScheduling">
            <invoke name="InitiateProductionScheduling"...
                     operation="requestProductionScheduling" inputVariable="PO" />
            <invoke name="CompleteProductionScheduling"...
                    operation="sendShippingSchedule" inputVariable="shippingSchedule"/>
        </sequence>
    </flow>
    <reply name="ReplyInvoice"... operation="sendPurchaseOrder" variable="Invoice"/>
</sequence>
<process/>
```

Fig. 5. The XML description of the Purchase Order process

composition designer. A compensation handler can be associated to the fault handler, it starts from the erroneous process itself to undo some steps that have already been completed and return the control back at the identified checkpoints.

3.3 Case Study

The example used in the following sections are based on the well known case study of the *PurchaseOrder* BPEL specification presented in [27]. This example, issued from electronic commerce, describes a service that processes a purchase order.

On receiving the purchase order from a customer, the process initiates three paths concurrently: calculating the final price for the order, selecting a shipper, and scheduling the production and shipment for the order. When some processes are concurrent, they induce control and data dependencies between the three paths. In particular, the shipping price is required to finalize the price calculation, and the shipping date is required for the complete fulfilment schedule. When the three concurrent paths are completed, invoice processing can proceed and the invoice is sent to the customer.

Figure 5 shows how the public interface of the *PurchaseOrderProcess* process is described by the *PurchaseOrder* Web service. This service publishes the *sendPurchaseOrder* operation that defines the received message (*POMessage* message) containing the customer and the purchase order information, and the replied message (*InvMessage* message) containing the invoice.

The behavior of the *PurchaseOrderProcess* process is described as a sequence of *ReceiveOrder*, *PurchaseOrderProcessing* and *ReplyInvoice* activities. *PurchaseOrderProcessing* is itself decomposed as a flow of *ProductionScheduling*, *ComputePrice* and *ArrangeLogistics* activities. *ProductionScheduling* is a sequence of *InitiateProductionScheduling* and *CompleteProductionScheduling* activities. *ComputePrice* is a sequence of *InitiatePriceCalculation*, *CompletePriceCalculation* and finally *ReceiveInvoice* activities. *ArrangeLogistics* is a sequence of *AssignCustomerInfo*, *RequestShipping* and *ReceiveSchedule* activities.

3.4 Decomposition Operator of BPEL

Structured activities represent the mechanism used by the BPEL language to describe the behaviour of each service composition operator in a BPEL process. A structured activity monitors and/or defines the execution order of activities a BPEL process is composed of. Thus, a structured activity can be decomposed into sub-activities (decomposition operation), which themselves can be decomposed into other sub-activities. This decomposition process can be repeated until simple activities are reached.

On figure 6, the decomposition of structured activities into other structured activities or simple ones is materialized by dashed vertical lines. The same figure shows how a BPEL process behaviour is described after decomposition. Initially, the $bpel_1$ process contains one structured activity. When a $bpel_i$ process contains one or more structured activities, they are decomposed into others activities and

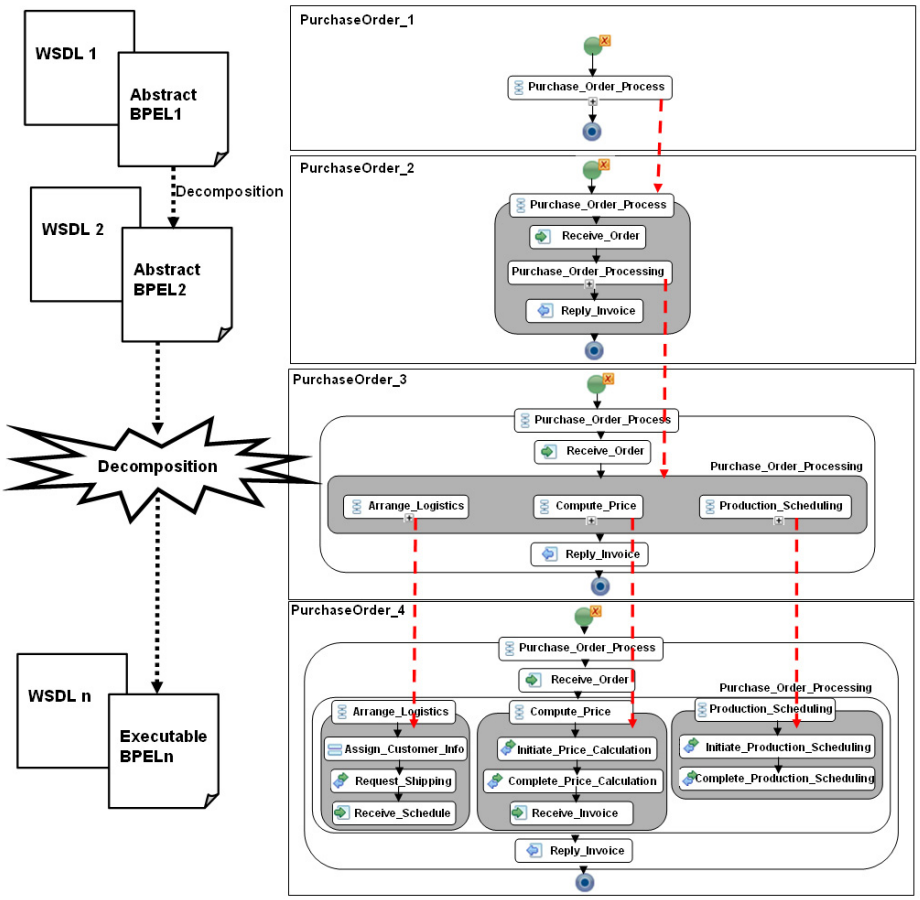


Fig. 6. Decomposition operator of BPEL applied to the Purchase Order process

we get another $bpel_i$ process with more detail in behaviour. As stated above, the decomposition process stops when no structured activity remains in the process description.

When designing the web services composition corresponding to this case study, graphical editors tools [13] are set up by the designers. These tools offer two design capabilities.

1. *A one shot web services composition description.* Here the designer produces the whole services composition in a single graphical diagram.
2. *A stepwise web services composition design.* In this case, the designer uses the decomposition operator offered by BPEL and encoded in graphical BPEL editors. This operator makes it possible to incrementally introduce more concrete services composing more abstract ones. Figure 6 shows a sequence of four decomposed BPEL models namely *PurchaseOrder_1*, *PurchaseOrder_2*, *PurchaseOrder_3* and *PurchaseOrder_4*.

Nowadays, it is well accepted that formal methods are useful to establish the relevant properties at each step. The association of refinement and decomposition would help to ensure the correctness of this decomposition. We claim that the Event-B method is a good candidate to bypass these insufficiencies.

3.5 BPEL and the Event-B Semantics

The study of the BPEL language and the standard describing its semantics [27] has revealed several links between this language and Event-B. In this section, we discuss the structural and the behavioural links.

Structural link. A BPEL description contains two main parts: the first part concerns data type, messages and operations description used by the web services that implement the composed service. This part is a static part of BPEL, it is described in WSDL and XSD files. The second part deals with variables description handled by the BPEL process, process behaviour description and different fault and events handlers associated with the described BPEL process. This part is a dynamic part of BPEL described in a BPEL file.

For the Event-B method, a model consists of two main components, Context and Machine. A Context contains abstract and enumerated sets definitions, constants declarations and properties related to the constants. This is a static part of an Event-B model. A Machine contains variables declarations, invariant properties and events that describe the model behaviour. This is a dynamic part of an Event-B model.

Behavioural link. BPEL describes a composed service as a process. The behaviour of this process is described himself by a main structured activity. This activity is interpreted as a state transition system. The activities controlled by the main activity represent the transitions. The messages and the data, described by a *variables* element in BPEL, represent the state of a BPEL process. The execution of the BPEL activities has as effect, to modify the messages contents and the state variables.

In the Event-B method, a Machine component encodes a state transition system where the state of the model is formalized in the VARIABLES clause, and the transitions are formalized by the events of the Event-B Machine. The events are guarded and the triggering of an event has an effect on state variables.

From this interpretation, formal modelling of BPEL by Event-B is based on the observation that a BPEL process is interpreted as a state transition system. A state is represented in both languages by a *variables* element in BPEL and by the VARIABLES clause in Event-B. The various activities of BPEL represent the transitions. They are formalized by the events of the EVENTS clause in Event-B.

4 From BPEL Process to Event-B Model

Our proposal consists in formally expressing the semantics of a BPEL process by an Event-B model in order to check the relevant properties defined in the

Event-B models by the services designers. A formal representation of a BPEL description by an Event-B model is driven by the BPEL process structure. Each BPEL element has a corresponding interpretation in Event-B. In the proposed interpretation rules, a *one-to-one* link between each BPEL element and its corresponding Event-B interpretation is guaranteed. As a result, when a proof obligation is generated in the Event-B model, it is uniquely linked to the associated BPEL element. The identification of the BPEL code part that generates this proof obligation becomes possible. This characteristic is important because it links the Event-B modelling directly in the BPEL description.

According to the separation of static and dynamic parts in BPEL description, the interpretation of a BPEL description by an Event-B model is obtained in two steps.

- The first step interprets the WSDL code that describes the various web services, their data types and operations into the different data types and functions offered by Event-B. This part is encoded in the Context of an Event-B model which is devoted to the description of the different data types and sets required for the definition of the Event-B model. Most of the related approaches omit this part. It is abstracted to reduce the size of the explored graph (Static part).
- The second step concerns the description of the composition of the activities appearing in a BPEL description. They are formalized as events. This composition is controlled by the synchronization of the events in the Event-B Machine (Dynamic part).

The process associated to these two steps is presented in the following subsections.

4.1 Formal Modelling of WSDL Service Description by an Event-B Context

A WSDL specification usually contains the declaration of data types, messages, port types (the profile of supported operations), bindings and services. Since a BPEL process references only the data types, the messages and the operations of the port types, solely the rules for interpreting these elements by an Event-B Context are defined.

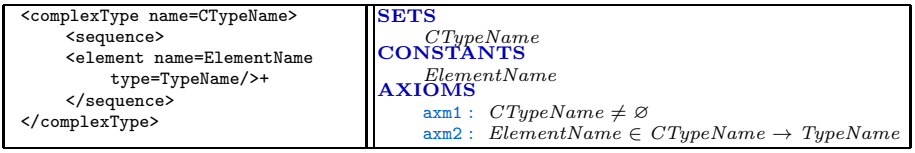


Fig. 7. An Event-B formalization of an XML complexType

Data Type. We chose to present the most often used constructor in the data type and messages declarations manipulated by a WSDL Web services definition. We have treated the sequence of elements in the case of a *complexType* XML definition. The XML *complexType* is represented by an abstract set and each *element* of a *complexType* is represented by a functional relation corresponding to the template $ElementName \in CTypeName \rightarrow TypeName$ (figure 7).

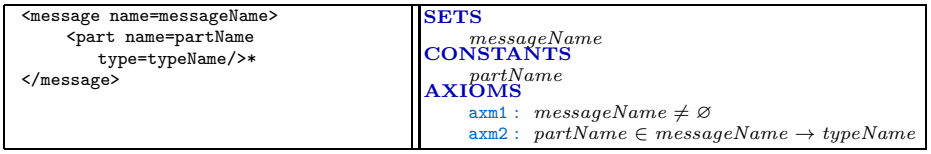


Fig. 8. An Event-B formalization of a WSDL message type

Message Type. The WSDL *message* element is formalized by an Event-B abstract set. Each *part* attribute of a *message* is represented by a functional relation corresponding to the template $partName \in messageName \rightarrow typeName$ from the *message* type to the part *type* (figure 8).

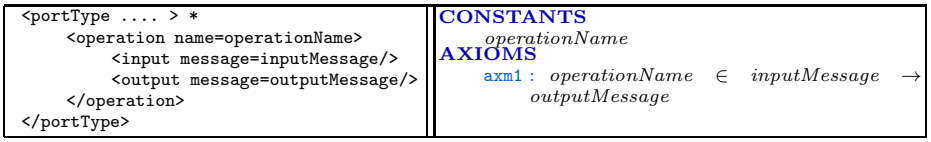


Fig. 9. An Event-B formalization of a WSDL operation type

Operation Profile. Each *operation* of a *portType* is represented by a functional relation corresponding to the template $operationName \in inputMessage \rightarrow outputMessage$ from the message type of the *input* element to the message type of the *output* element (figure 9).

The *portType* element is not represented in the Context, because the composition process refers only to the operation offered by the port. As a consequence, only the operation is represented. Notice that the rest of the WSDL elements is not used in BPEL.

Application to the Case Study. For example, consider the *purchaseOrderType* type shown in figure 10. A set named *purchaseOrderType* is defined in the SETS clause. For the *CID* element, describing the customer identifier, the *CID* set is introduced in the AXIOMS clause as a functional relation (axiom *axm1* in Figure 10). The *POMessage* message element is formalized by a *POMessage* abstract set defined in the SETS clause. For the *customerInfo* attribute of the *POMessage* message, the *customerInfoPOMessage* set is introduced in the AXIOMS clause as a functional relation (axiom *axm5* in Figure 10). Finally, the

sendPurchaseOrder operation of the *purchaseOrderPT portType* is encoded by the functional relation described in the axiom *axm9*.

The previous translation rules define the static part of an Event-B model, namely the Context. It encodes the whole definitions of WSDL description relevant for defining Event-B models. This part may be extended with AXIOMS and THEOREMS that express properties on the defined data elements. The formalization of this part is relevant for checking data oriented properties.

```
<complexType name="purchaseOrderType">
    <element name="CID" type="int"/>
    <element name="order" type="int"/>
    ...
</complexType>
...
<message name="POMessage">
    <part name="customerInfo"
        type="customerInfoType"/>
    <part name="purchaseOrder"
        type="purchaseOrderType"/>
</message>
...
<portType name="purchaseOrderPT">
    <operation name="sendPurchaseOrder">
        <input message="POMessage"/>
        <output message="InvMessage"/>
    </operation>
</portType>
```

CONTEXT *PurchaseOrderServices*
SETS
POMessage
CustomerInfoType
PurchaseOrderType
...
CONSTANTS
CID order ... customerInfoPOMessage
purchaseOrderPOMessage ...
sendPurchaseOrder
AXIOMS
axm1 : $CID \in PurchaseOrderType \rightarrow Z$
axm2 : $order \in PurchaseOrderType \rightarrow Z$
axm : ...
axm5 : $customerInfoPOMessage \in POMessage \rightarrow CustomerInfoType$
axm6 : $purchaseOrderPOMessage \in POMessage \rightarrow PurchaseOrderType$
axm : ...
axm9 : $sendPurchaseOrder \in POMessage \rightarrow InvMessage$
axm : ...

Fig. 10. An Event-B Context for complex Types, messages and operations

4.2 Formal Modelling of BPEL Variables and Activities by Event-B Variables and Events

This section addresses the description of the dynamic part of a BPEL process definition. The formalization process is inductively defined on the structure of the BPEL definition. Each BPEL variable corresponds to a state variable of the Event-B model in the VARIABLES clause, each simple activity becomes an event of the Event-B model and each structured activity is translated to a specific events construction. The following interpretation rules are set-up.

Variables. The BPEL variable element is represented by a variable in the VARIABLES clause in an Event-B Machine. This variable is typed in the INVARIANTS clause using *type* BPEL attribute. Each *variableName* is represented by a set of size one, initialized to the empty set (figure 11). This interpretation makes it possible to check if a state variable is empty or not (case of message for example).

For example, the *POMessage* message of the *PO* variable is represented by the invariants *inv1* and *inv2* in Figure 12.

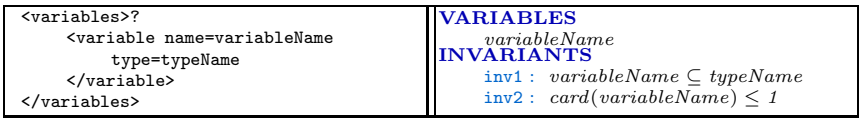


Fig. 11. An Event-B formalization of a BPEL variables

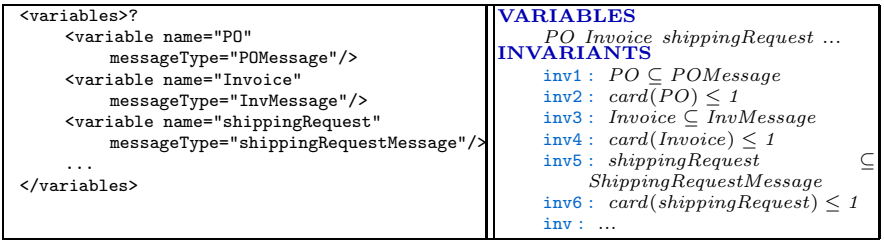


Fig. 12. BPEL variables modelled in the VARIABLES clause

Simple Activities. Each BPEL *simple activity* [27] is interpreted by a specific event in the EVENTS clause of the Event-B Machine. The *invoke* activity is used to invoke a Web service, the *receive* activity is used to receive a message from outside environment of the process, the *reply* activity is used to return an answer to a service partner, the *assign* activity is used to change the content of variables, the *wait* activity is used as a temporization to wait for a given period of time or up to a given point in time, and the *terminate* activity is used to end the process.

In next paragraphs, we give three Event-B models that formalize the *invoke*, *receive* and *reply* activities. The rest of simple activities corresponds to a simple Event-B action: an affectation action formalizes the *assign* activity and an action that sets the value of the Event-B *variant* to zero formalizes the *terminate* activity. Regarding the *wait* activity, it can be formalized by adding a decreasing integer variable encoding a discrete time period.

The *invoke* activity, named $activityName$, invokes an $operationName$ operation of a web service with $inputVariableName$ variable as input and $outputVariableName$ variable as output. This activity is formalized by the $activityName$ event (figure 13). The guard of the $activityName$ event expresses the guard to be checked for using an operation: the input message must be non-empty ($grd1$).

The *receive* activity, named $activityName$, receives a message, stored in the $variableName$ variable, sent by a partner web service to execute the $operationName$ operation. This activity is interpreted by the $activityName$ event (figure 14). The guard of the $activityName$ event expresses the conditions to be checked before receiving the message: the received message belongs to the domain of the $operationName$ function ($grd1$).

The *reply* activity, named $activityName$, sends a message, stored in the $variableName$ variable, to a partner web service after executing the $operationName$ operation. This activity is interpreted by the $activityName$ event

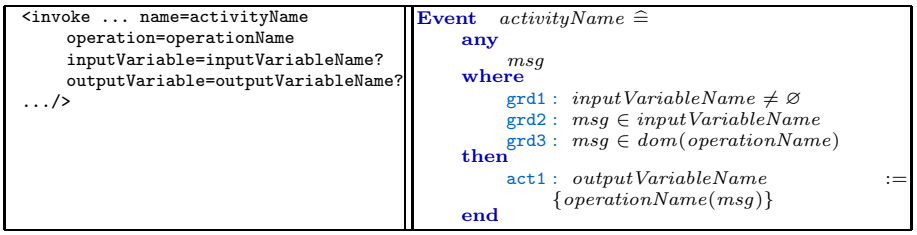


Fig. 13. An Event-B formalization of a BPEL invoke activity

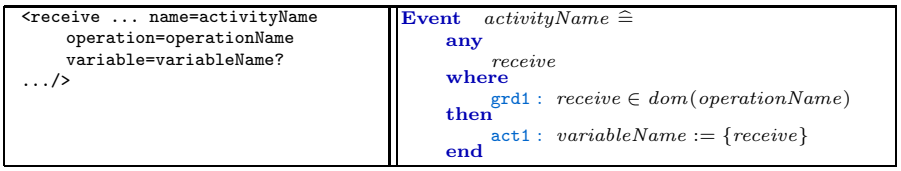


Fig. 14. An Event-B formalization of a BPEL receive activity

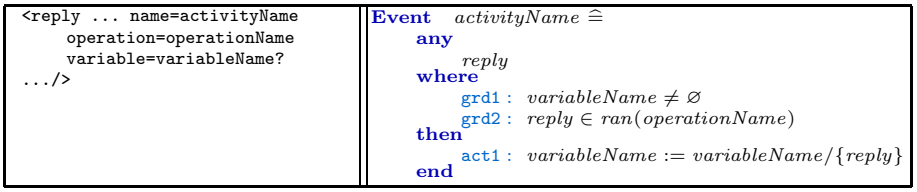


Fig. 15. An Event-B formalization of a BPEL reply activity

(figure 15). The guard of the *activityName* event expresses the conditions to be checked before sending the message: the sent message must be non-empty ($grd1$) and belonging to the range of the *operationName* function ($grd2$).

In all these cases, the status of the obtained event depends on the structured activity that composes the corresponding activity. This is addressed in the following section.

Structured Activities. Each BPEL structured activity (*flow*, *sequence*, *foreach*, *if then else*...) is modelled by an Event-B construction which encodes the corresponding composition operator (see Table 1). modelling composition operations in Event-B follows the formal modelling rules formally defined in [5]. We have introduced the use of explicit decreasing variant for defining the control flow.

To illustrate our approach, we show the Event-B templates associated to sequence, concurrency, choice and iteration of table 1.

Table 1. The composition operators corresponding to the structured activities

BPEL activity	Operator name	Operator symbol
sequence	Sequence	>> or ;
while repeat until forEach	Iteration	$*$
flow	Concurrency	$\|\|$
if then else	Choice	$[]$

Sequence. Let us consider the action A_0 corresponding to the activation in sequence of A_1 and A_2 actions. The Event-B Machine formalizing the sequence operator contains three events evt_0, evt_1 and evt_2 formalizing the three actions A_0, A_1 and A_2 (figure 16). This Machine uses a variant expressed by the $varSeq$ variable initialized to the value 2. The evt_1 and evt_2 events are declared "*convergent*" and once the guard of evt_1 is evaluated to "*true*" ($varSeq = 2 \wedge G_1(var_1)$), the event is fired and the variant is decreased (the value of $varSeq$ is set to 1). The evt_2 event can be fired after evt_1 event, when its guard is evaluated to true ($varSeq = 1 \wedge G_2(var_1)$), and the value of $varSeq$ is set to 0. The evt_0 event ends the sequence operation of A_1 and A_2 actions ($varSeq = 0$).

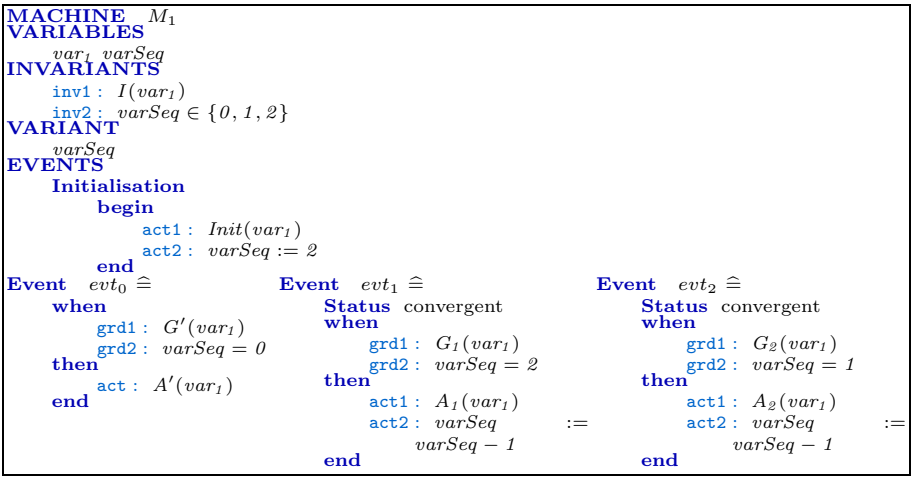


Fig. 16. Encoding sequence operator in Event-B

Concurrency. Let us consider the activation of A_0 action as concurrent activation of A_1 and A_2 actions. The associated semantics is interleaving, imposes to describe all the possible behaviours (all the possible traces). It uses the interleaving underlying Event-B semantics. Three events evt_0, evt_1 and evt_2 corresponding to the three actions A_0, A_1 and A_2 are defined in the Event-B Machine formalizing the parallel operator (figure 17). This Machine uses a variant expressed by the sum of $varPar_1$ and $varPar_2$ variables that are both initialized to the

value 1. evt_1 and evt_2 events are declared "*convergent*" and if the two events evt_1 and evt_2 have their guard evaluated to "*true*" ($varPar_1 = 1 \wedge G_1(var_1)$ and $varPar_2 = 1 \wedge G_2(var_1)$), they are fired in parallel in an interleaving manner and the substitutions $A_1(var_1)$ and $A_2(var_1)$ are performed and the value of the variant is set to the value 0 (the values of $varPar_1$ and $varPar_2$ are set to 0). The evt_0 event ends the parallel operation of evt_1 and evt_2 events.

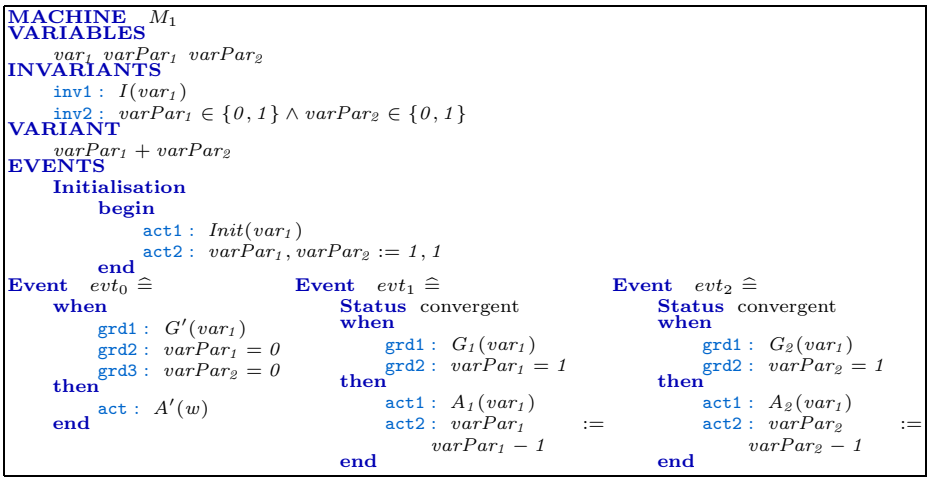


Fig. 17. Encoding concurrency or parallel operator in Event-B

Choice. Let us consider action A_0 corresponding to the non deterministic choice between A_1 and A_2 actions (either A_1 and A_2 is fired). The Event-B model formalizing the choice operator contains three events evt_0, evt_1 and evt_2 formalizing the three actions A_0, A_1 and A_2 (figure 18). This Machine uses a variant expressed by the $varCho$ variable initialized arbitrarily to either 1 or 2 ($varCho :\in \{1, 2\}$). The evt_1 and evt_2 events are declared "*convergent*" and according to the guard value of each event, one of evt_1 and evt_2 events is fired. Each event decreases immediately the variant to value 0 forbidding the other events to be fired. The evt_0 event ends the firing of choice operation between evt_1 and evt_2 events ($varCho = 0$).

Iteration. Let us consider the action A_0 as a loop execution of the A_1 action. The principle of encoding a loop in Event-B consists in firing, N times (N being an arbitrary natural number), the event corresponding to the action A_1. The Event-B model formalizing the loop operator contains two events evt_0 and evt_1 formalizing the actions A_0 and A_1 (the body of the loop) (figure 19). This Machine uses a variant expressed by the $varLoop$ variable initialized to the value N corresponding to an arbitrary number of iterations. The evt_1 event is declared "convergent" and according to the initial value of the $varLoop$ variable, the evt_1 event is fired N times. At each iteration, the evt_1 event decreases the variant

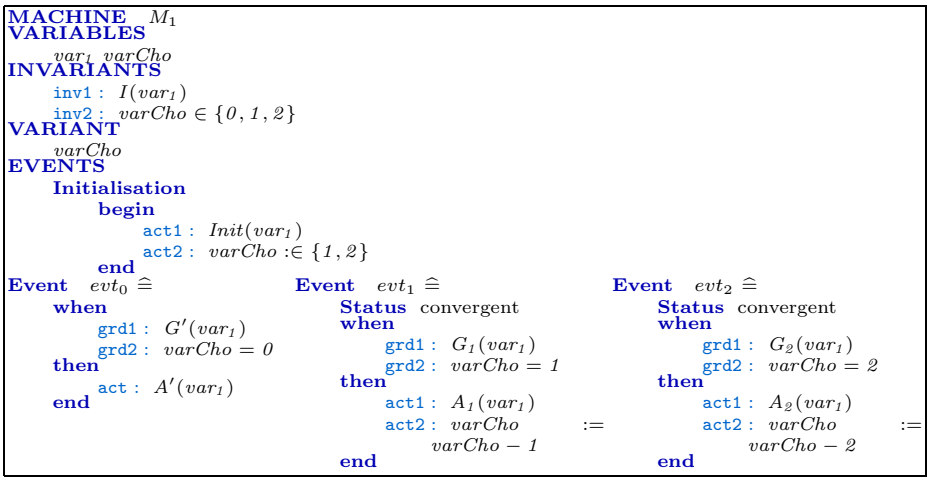


Fig. 18. Encoding choice operator in Event-B

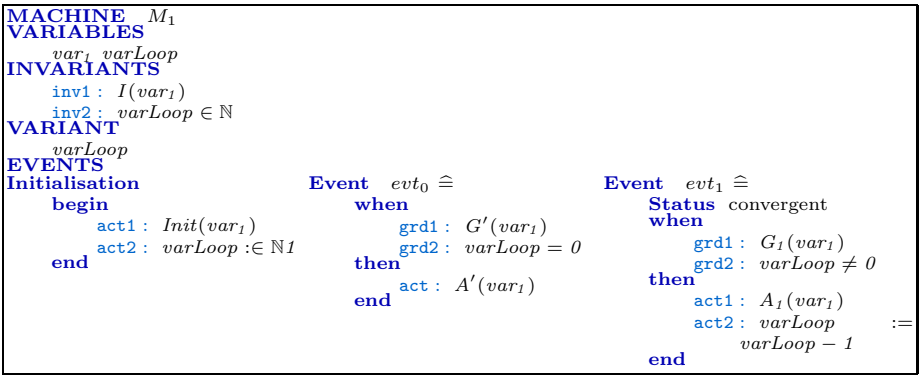


Fig. 19. Encoding loop operator in Event-B

until the value 0. When the loop terminates ($varLoop = 0$), the evt_0 event is fired, it ends the loop operator.

The initial value of the variant is arbitrary fixed by the $:\in$ operator. The advantage of such an approach is the possibility to encode an arbitrary number of loop steps without increasing the complexity of the proof process. Compared to model checking techniques, increasing the number of loop steps may lead to the combinatorial explosion problem.

In Figure 20, we show the Event-B model for a sequence of three simple BPEL activities described in Figure 5: first the *invoke* activity formalized by the *initiatePriceCalculation* event, second the other *invoke* activity formalized by the *sendShippingPrice* event and last the *receive* activity formalized by the *sendInvoice* event. These events are synchronized by a sequence through the

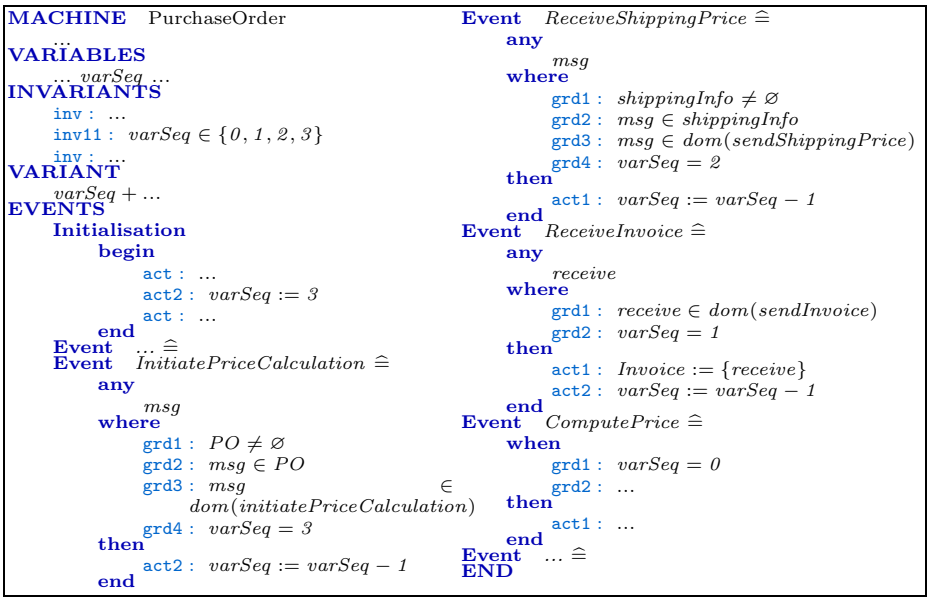


Fig. 20. A sequence of BPEL activities in Event-B

varSeq variable playing the role of a variant on the *PurchaseOrder* Machine. The *ComputePrice* event ends the sequence structured activity.

5 A Refinement Based Methodology

The previous section addressed the interpretation of BPEL descriptions to Event-B models. This interpretation process is qualified as *horizontal interpretation* produces an Event-B model from any BPEL description whatever is the reached level of the design on the BPEL side. This approach may lead to complex Event-B models with complex proof obligations requiring interactive proofs. It neither takes into account the stepwise decomposition, shown in figure 6, supported by the BPEL descriptions and encoded by the decomposition operator, nor the refinement capability offered by the Event-B method.

5.1 Methodology: Vertical Decomposition

An incremental construction of the BPEL specification using process decomposition operations (figure 6) will help to build reusable BPEL specifications and less complex Event-B models. Our claim is to encode each BPEL decomposition operation (adding new activities) by a refinement in Event-B (adding new events). This decomposition corresponds to a vertical decomposition.

To define this vertical decomposition process, we use the two following definitions.

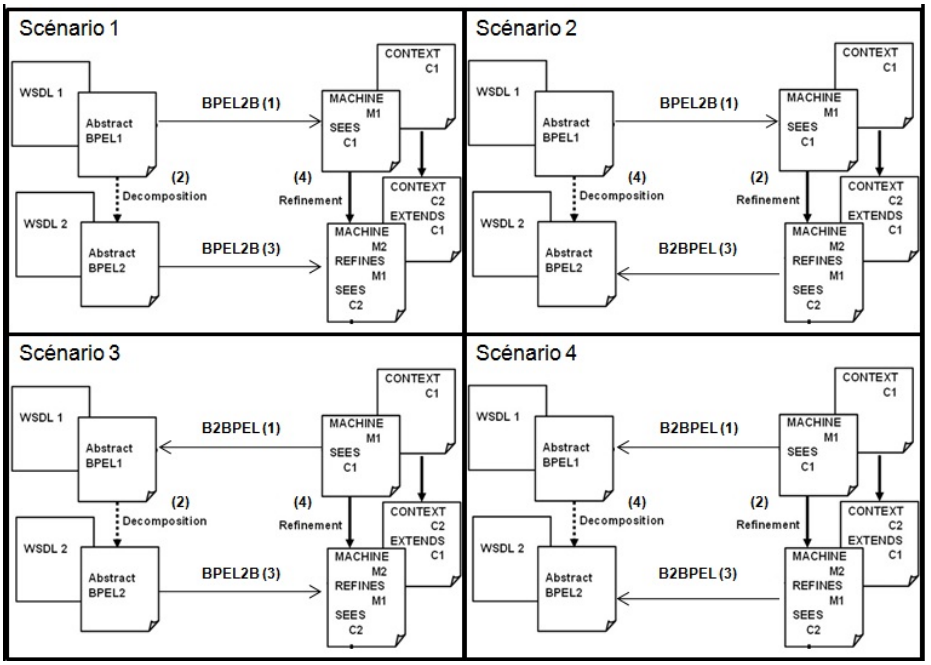


Fig. 21. Different scenarios for vertical decomposition of BPEL description

1. When a $bpel_i$ BPEL process contains structured activities, they are decomposed into sub-activities and we get by this operation a $bpel_{i+1}$ new BPEL process. Each $bpel_i$ BPEL process interacts with a set of partners web services described in a $wsdl_i$ WSDL document.
2. Each m_i Event-B Machine imports a c_i Event-B Context. When m_{i+1} Machine refines a m_i Machine, the c_i Context is extended by the c_{i+1} Context.

As depicted in figure 21, the beginning of each scenario is performed from a $bpel_1$ BPEL process containing a main structured activity or from a m_1 Machine containing an Event-B event. Four scenarios are associated with this vertical decomposition process.

1. *Scenario* 1
   - The $bpel_i$ process is translated into a m_i Machine and the $wsdl_i$ document is translated into a c_i Context.
   - The $bpel_{i+1}$ process is obtained by decomposition of the $bpel_i$ process. The web services invoked by the $bpel_{i+1}$ process are described in the $wsdl_{i+1}$ document.
   - The $bpel_{i+1}$ process is translated into a m_{i+1} Machine and the $wsdl_{i+1}$ document is translated into a c_{i+1} Context.
   - The obtained m_{i+1} Machine refines the m_i Machine and the c_{i+1} Context extends the c_i Context.

2. *Scenario* 2
   - The $bpel_i$ process is translated into a m_i Machine and the $wsdl_i$ document is translated into a c_i Context.
   - The m_i Machine is refined by the m_{i+1} Machine and the c_i Context is extended by the c_{i+1} Context by adding definitions of the services introduced by the MACHINE m_{i+1}.
   - The $bpel_{i+1}$ process is obtained from the m_{i+1} Machine and the $wsdl_{i+1}$ document is obtained from the c_{i+1} Context.
   - The obtained $bpel_{i+1}$ process decomposes the $bpel_i$ process and it invokes the web services described in the $wsdl_{i+1}$ document.
3. *Scenario* 3
   - The $bpel_i$ process is obtained from the m_i Machine and the $wsdl_i$ document is obtained from the c_i Context.
   - The obtained $bpel_{i+1}$ process decomposes the $bpel_i$ process and it invokes the web services described in the $wsdl_{i+1}$ document.
   - The $bpel_{i+1}$ process is translated into a m_{i+1} Machine and the $wsdl_{i+1}$ document is translated into a c_{i+1} Context.
   - The obtained m_{i+1} Machine refines the m_i Machine and the c_{i+1} Context extends the c_i Context.
4. *Scenario* 4
   - The $bpel_i$ process is obtained from the m_i Machine and the $wsdl_i$ document is obtained from the c_i Context.
   - The m_i Machine is refined by the m_{i+1} Machine and the c_i Context is extended by the c_{i+1} Context by adding definitions of the services introduced by the MACHINE m_{i+1}.
   - The $bpel_{i+1}$ process is obtained from the m_{i+1} Machine and the $wsdl_{i+1}$ document is obtained from the c_{i+1} Context.
   - The obtained $bpel_{i+1}$ process decomposes the $bpel_i$ process and it invokes the web services described in the $wsdl_{i+1}$ document.

So, we obtain on the one hand a sequence of abstract BPEL specifications, one being the decomposition of the other, and on the other hand a sequence of proved Event-B refinements corresponding to the abstract BPEL specification. Then a formal stepwise refinement of a BPEL specification is obtained. From the methodological point of view, the last obtained BPEL process description is the one that is deployed for execution by the orchestrator in charge of execution (see figure 6).

The interpretation of an Event-B model into a BPEL description is not considered in our work. Therefore, the study of scenarios 2, 3 and 4 is not discussed in this paper. Defining and writing generalized rules for Event-B to generate BPEL description from Event-B models, makes it possible to support scenarios 2, 3 and 4. The study of these scenarios is left for of future work. In the following, only the scenario 1 is taken into account in the vertical decomposition process.

5.2 The Application of Scenario 1 to the Case Study

When applied to our case study, the proposed methodology leads to a development of a sequence of 4 Event-B machines. Each one refining the previous

one. As depicted on the right hand side of figure 22, the Event-B Machines *PurchaseOrder_1*, *PurchaseOrder_2*, *PurchaseOrder_3* and *PurchaseOrder_4* define the development of our case study. In order to show the benefits of this approach, we have chosen to comment below the machines obtained after the third refinement.

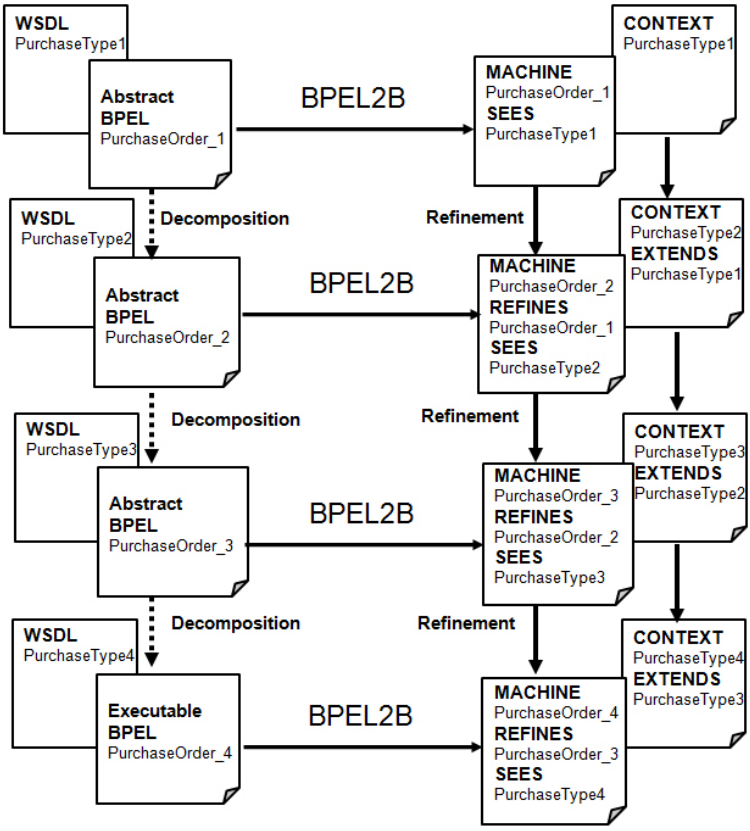


Fig. 22. The application of the scenario 1 to the Purchase order case study

Figure 23 shows part of the *PurchaseOrder_3* Machine obtained after refinement. The event *ComputePrice* computes the total amount of the invoice to be produced at the end of shipping. This machine is completed by the relevant resources needed to ensure the correct computation of the total amount of the invoice. The invariants *inv*17, *inv*18 and *inv*19 and the action *act*1 are interactively added by the developer. There is no way to generate them from the BPEL code since they are not available.

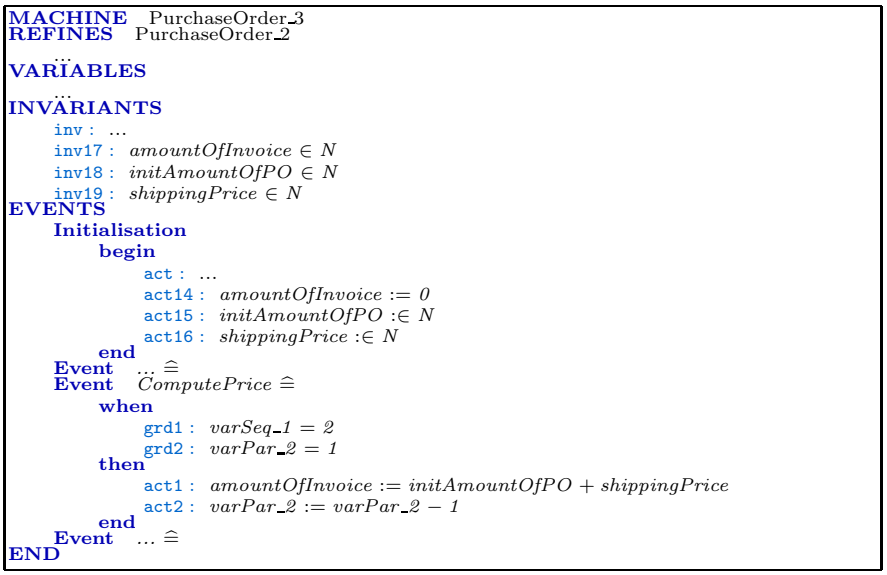


Fig. 23. A PurchaseOrder_3 Event-B Machine

Machine *PurchaseOrder_4* refines the Machine *PurchaseOrder_3* as shown in figure 24. Again, some information are completed by the developer in order to ensure the correctness of the development. The concrete variables SP, $IAOP$ and AOI of the invariant $inv11$ together with the gluing invariant $inv12$ and the guards $grd5$ are introduced. The introduction of the $grd5$ guards of the *InitiatePriceCalculation* and *ReceiveShippingPrice* events, ensures that the concrete variables introduced for defining the gluing invariant are correctly related to the exchanged messages produced by the functions carried by the specific services.

The actions modifying these variables are also defined by introducing the $act2$ and $act3$ in the *InitiatePriceCalculation* and *ReceiveShippingPrice* events. They show how the variables evolve. The sequencing of the events *InitiatePriceCalculation*, *ReceiveShippingPrice* and *ReceiveInvoice* supplying the result $amo- untOfInvoice$ to the *ComputePrice* refined event remains automatically produced.

6 Verification of Services Properties

When the Event-B models formalizing a BPEL description are obtained, we have seen in the previous section that they may be enriched by the relevant properties that formalize the user requirements and the soundness of the BPEL defined process. In Event-B, these properties are defined in the AXIOMS, INVARIANTS and THEOREMS clauses. Preconditions and guards are added to define the correct event triggering. Our work considers arbitrary sets of values

MACHINE PurchaseOrder_4
REFINES PurchaseOrder_3
...
INVARIANTS
inv : ...
inv10 : $varSeq\_12 \in \{0, 1, 2, 3\}$
inv11 : $SP \in N \land IAOP \in N \land AOI \in N$
inv12 : $AOI + IAOP + SP = initAmountOfPO + shippingPrice$
EVENTS
Initialisation
begin
act : ...
act13 : $varSeq\_12 := 3$
act14 : $amountOfInvoice, AOI := 0, 0$
act15 : $initAmountOfPO, IAOP :\mid (...IAOP = initAmountOfPO)$
act16 : $shippingPrice, SP :\mid (...SP = shippingPrice)$
end
Event ... $\widehat{=}$
Event *InitiatePriceCalculation* $\widehat{=}$
any
msg
where
grd1 : $PO \neq \varnothing$
grd2 : $msg \in PO$
grd3 : $msg \in dom(initiatePriceCalculation)$
grd4 : $varSeq\_12 = 3$
grd5 : $IAOP = Price(msg)$
then
act1 : $varSeq\_12 := varSeq\_12 - 1$
act2 : $AOI := AOI + IAOP$
act3 : $IAOP := 0$
end
Event *ReceiveShippingPrice* $\widehat{=}$
any
msg
where
grd1 : $shippingInfo \neq \varnothing$
grd2 : $msg \in shippingInfo$
grd3 : $msg \in dom(sendShippingPrice)$
grd4 : $varSeq\_12 = 2$
grd5 : $SP = shippingPrice(msg)$
then
act1 : $varSeq\_12 := varSeq\_12 - 1$
act2 : $AOI := AOI + SP$
act3 : $SP := 0$
end
Event *ReceiveInvoice* $\widehat{=}$
any
receive
where
grd1 : $receive \in dom(sendInvoice)$
grd2 : $varSeq\_12 = 1$
grd3 : $AOI = amount(receive)$
then
act1 : $Invoice := \{receive\}$
act2 : $varSeq\_12 := varSeq\_12 - 1$
end
Event *ComputePrice* $\widehat{=}$
refines *ComputePrice*
when
grd1 : $varSeq\_12 = 0$
grd2 : ...
then
act : ...
act3 : $amountOfInvoice := AOI$
end
END

Fig. 24. A PurchaseOrder_4 Event-B Machine

for parameters defining their types. There is no abstraction related to the parameters, preconditions, postconditions nor join operations. The expressions are represented as they appear in BPEL.

The proof based approach we propose does not suffer from the growing number of explored states. More precisely, regarding the formal verification of properties, our contribution is summarized in the following points.

- *BPEL type control.* Static properties are described in the Context of the Event-B model. They concern the description of services, messages and their corresponding types (WSDL part). Event-B ensures a correct description of the types and function composition.
- *Orchestration and services composition.* Dynamic properties are described in the Machine of an Event-B model, they concern the variables (messages exchanged between services) and the BPEL process behaviours (BPEL part). The introduction of variants guarantees the correct services triggering order and message passing.
- *Deadlock freeness.* It is always possible to trigger at least one event. This property is ensured by asserting (in the THEOREMS clause) that the disjunction of all the abstract events guards implies the disjunction of all the concrete events guards. It ensures that if at least one guard of the abstract Event-B model is true, then at least one guard of the refined model is true.
- *No LiveLock.* A decreasing variant is introduced for the purpose of the definition of each refinement corresponding to the encoding of a composition operator. When this variant reaches the value 0, another event may be triggered.
- *Pre-condition for calling a service operation: input message is not empty.* In the orchestration tools, the condition for triggering a BPEL activity is the correct reception of the message used by this activity. Our representation of the call of a service operation takes into account this condition in the events guards corresponding to this activity and an invariant guarantees the existence of this state.
- *Data transformation.* Data properties are expressed in the in the INVARIANTS and AXIOMS clauses. They are checked for each triggered event. This ensures a correct manipulation and interpretation of data and of messages exchanged between all the participants (partners).
- *Transactional properties.* When modelling fault and compensation handlers by a set of events, it becomes possible to model and check properties related to transactional web services. The idea was discussed in [9] and we will extend it in our future work by defining a methodology for designing a transactional BPEL process.

Table 2 summarizes the number of PO generated by the RODIN platform [29] for the Event-B model associated to the *Purchase Order* process. For the *purchaseOrder* Machine, obtained from the *Purchase Order* process without decomposition/refinement, 67 POs were generated with 53 POs automatically proven by the prover of the RODIN platform and 14 POs required interaction with the

designer for performing an interactive proof. For the case of scenario 1 using decomposition/refinement, 67 POs were generated with 56 POs automatically proven by the prover of the RODIN platform and 11 POs required interaction with the designer for performing an interactive proof.

Table 2. The obtained PO from the Event-B model of the *Purchase Order* process

Event-B Machine	**Total of PO**	**Automatic proof**	**Interactive proof**	**%Pr**
purchaseOrder	67	53	14	100
PurchaseOrder_1	0	0	0	100
PurchaseOrder_2	16	16	0	100
PurchaseOrder_3	12	9	3	100
PurchaseOrder_4	39	31	8	100
total	67	56	11	100

From the point of view of the number of proof obligations proved interactively in the case of Scenario 1, we record less interactive proof (11 against 14) with the decomposition and simplification of expressions of properties at different refinements. In addition, for the 11 POs proved interactively, 7 required a single proof step i.e. only 4 required proving efforts.

But, one of the major interests of the proposed methodology is error reporting. Indeed, when an Event-B model cannot be proved due to an erroneous BPEL design and/or decomposition, it is possible to report, on the BPEL code, the occurred error issued from the Event-B verification step, in the concerned activity. This functionality is very helpful for designers that are usually non specialists of formal methods.

Since each BPEL concept is attached to a single Event-B element preserving the same name as in the original BPEL design, it becomes possible to identify, localize and visualize the BPEL concept that corresponds to the Event-B element whose proof obligations cannot be discharged. Notice, that this reporting is only possible for the Event-B parts generated from the BPEL design.

This approach of tracing the proof obligations to the source BPEL code is very useful. it allows the developer to localize the possible errors in its developed service composition. Particularly, we have exploited this facility to identify transactional parts in web services composition. Briefly, the approach is based on the definition of transactional invariants. The transactional properties and the properties related to the consistencies of resources used by the BPEL process are expressed in the form of consistency invariants *(Consistency property)*. Once the enrichment of the generated Event B models is performed, POs are generated. Some of these POs, related to invariants involving the transactional properties, are unprovable because triggering some events separately violates the consistency invariants. Then, BPEL activities related to the events source of these improvable POs are detected and isolated in a BPEL scope element. our approach for handling transactional BPEL parts recommends to set up the mechanisms for fault and compensation handling to the scope element. As a

consequence, the execution of this part is isolated by the orchestration tools *(Isolation property)*, and at the same time consistency of the resources used by these activities is guaranteed. Transactional properties may require a re-design of the defined BPEL process. Due to space limitations, this approach is not presented in this paper.

7 BPEL2B Tool

With the aim to provide a complete tool for the web services compositions verification, we have integrated various plug-ins in a single platform. This platform is based on the Eclipse core and contains the WSDL and BPEL editors plug-ins [13], the BPEL2B [7] and B2EXPRESS [6] developed plugins and the various plugins of the RODIN platform [29]. Different views are offered by this platform (figure 25): WSDL and BPEL editors (a) to describe different web services and their orchestration graphically or using the XML syntax, BPEL2B plug-in (1,2) to interpret the WSDL/BPEL specifications by Event-B models (b), the different plug-ins of RODIN (c) to perform web services composition verification on the obtained Event-B model and the B2EXPRESS plug-in to animate the obtained Event-B model.

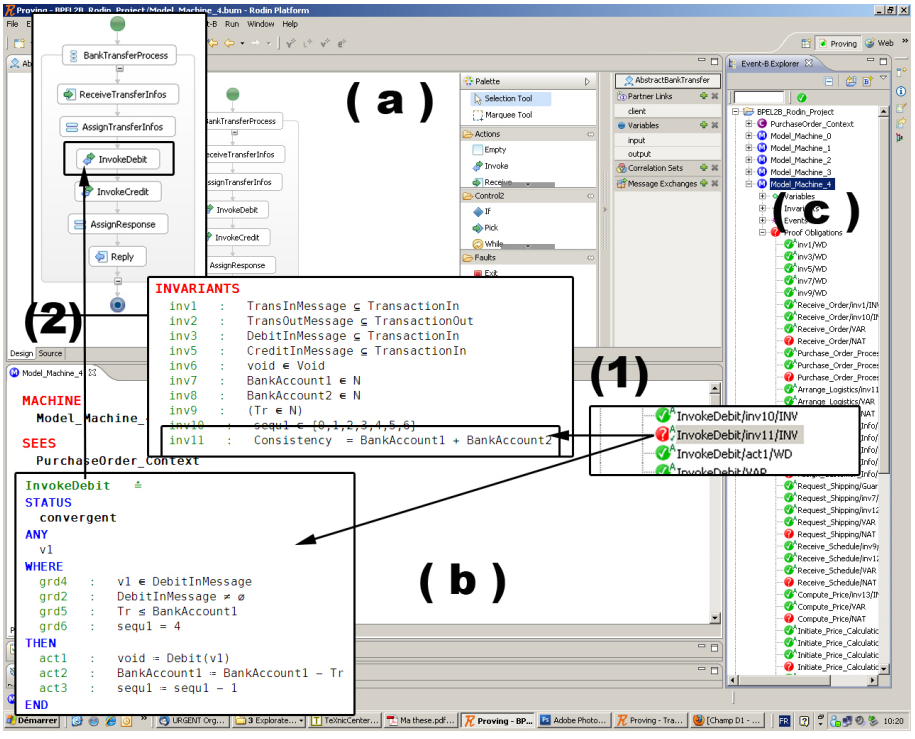


Fig. 25. The BPEL2B interface

The Event-B based approach, proposed for BPEL processes verification [8, 10], defines different interpretation rules from a BPEL description to an Event-B model. These rules are based on the interpretation of an element or an attribute of BPEL by a specific clause in an Event-B model. We have automated this interpretation process in the BPEL2B plug-in. This plug-in builds a RODIN project from WSDL and/or BPEL files. A RODIN project consists essentially of two types of components: Context and Machine. As described in section 4, the Context is obtained from the WSDL description and the Machine from the BPEL specification. The implementation of the transformation rules are based on the Event-B meta-model provided by RODIN platform and WSDL/BPEL meta model provided by the BPEL editor.

The transformation process has been encoded using JAVA and using a model transformation describing a generic transformation between both RODIN and BPEL meta-models. As a consequence, following a model driven engineering approach, the BPEL models, instances of the corresponding meta-model are automatically transformed to instances of the RODIN meta-model.

Moreover, this transformation process preserves a link between the source and target models, so that it is possible to report in the source BPEL model, the errors that may occur while proving the Event B model.

The RODIN platform uses as input the obtained RODIN project, containing both Machine and Context components. When the Event-B models, formalizing a BPEL description are obtained, they may be enriched by the relevant properties that formalize the user requirements and the soundness of the BPEL defined process (for example adding a gluing invariant). In Event-B, these properties are defined in the INVARIANTS and the THEOREMS clauses. The proof activity is performed using the prover of the RODIN platform.

8 Related Work

Various approaches have been proposed to model and to analyze web services and services compositions, especially formal modelling and verification of BPEL processes. In this section, we overview the formal approaches used for formal verification of services composition descriptions.

Petri nets were often used to model web service compositions. They have been set up by *Hinz et. al* [20] to encode BPEL processes and to check standard Petri nets properties, and some other properties formalized in CTL logic. *van der Aalst et. al* [1, 32] have defined specific Petri nets called workflow nets to check some properties like termination of a workflow net, and detection of unreachable nodes. Recently, *Lohmann* [22] has completed the BPEL semantics with Petri nets by encoding the new elements appeared in the version 2.0 of BPEL.

Classical transitions systems were also applied to specify web service compositions, especially BPEL processes. We quote the work of *Nakajima* [25, 26] who mapped a BPEL activity part to a finite automaton encoded in Promela with the model checker SPIN to analyze behavioral aspects and to detect potential deadlocks that may occur in the Promela description. In [23, 24], *Marconi et.*

al present the approach that translates a BPEL process to a set of state transition systems. These systems are composed in parallel and the resulting parallel composition is annotated with specific web services requirements and is given as input to a set of tools in charge of synthesizing web service compositions expressed in BPEL. FSP (Finite State Process) and the associated tool (LTSA) are used by *Foster et. al* [17–19] to check if a given web service compositions behaves like a Message Sequence Charts (MSC).

Some work used process algebra for processes and activities formalization. Indeed, *Salaun et. al* [30] show how BPEL processes are mapped to processes expressed by the LOTOS process algebra operations. The same authors in [31] applied their approach to CCS descriptions to model the activities part of a BPEL specification.

Abstract State Machines (ASM) have been used by *Farahbod et. al* [16] to model BPEL workflow descriptions. They take into account exchanged messages and some BPEL real time properties like timeouts. This work has been extended by *Fahland* [15] to model dead-path-elimination. *Borger et. al* [12] have also used ASM for modelling Workflow, specifically, BPMN process.

Other approaches proposed formal models for web service compositions and the BPEL processes. We did not mention them in the above summary due to space limitations. An overview of these approaches can be found in [11].

The whole related work outlined above has two major drawbacks. First, it does not address the description of the static part of BPEL available in the WSDL descriptions. This is due to the abstraction made by the majority of the applied formal methods. Indeed, most of these methods abstract parameters, exchanged messages, preconditions, postconditions and join conditions by Boolean. This abstraction is useful for model checking since it reduces the space explored during verification but it decreases the accuracy of the obtained formal BPEL model. Second, all these proposals translate a BPEL process to another formal model, without offering the capability to decompose BPEL precess. So, the decomposition operator offered by BPEL is never addressed by the used formal techniques. The resulting model is often complex to analyze. Thus, the efficiency of the model checking technique often used by existing approaches for checking BPEL processes properties is reduced.

Finally, these methods suffer from the absence of error reporting. Indeed, when errors are detected in the formal model, these approaches do not localize the source of the error on the original BPEL model.

Our work is proof oriented and translates the whole BPEL language and all its constructs into an Event-B model. All the Event-B models presented in this paper have been checked within the RODIN platform.

9 Conclusion

This paper presented a fully formalized and integrated method for designing correct web services compositions expressed in the BPEL description language. Our contribution addresses both technical and methodological points of view.

From the technical point of view, this approach proposes to encode each BPEL services composition description by an Event-B model in which relevant properties related to deadlock or livelock, data interpretation, messages consistence or transactions are modelled. In most of the cases, establishing these properties requires an enrichment (model annotation) of the Event-B models by the relevant information that are not available in the original BPEL description (no specific resource for encoding such properties is available in the BPEL language) but which are extracted by the developer from the informal requirements.

From the methodological point of view, our approach is top-down. It follows the decomposition process preconized by BPEL. It suggests to encode the decomposition relationship available in BPEL. As a result, the refinement chain of Event-B models is structurally linked to the decomposition process offered by the BPEL description language. The interest of this approach is double. On the BPEL side it offers a stepwise design approach while it eases the proof activity on the Event-B side since the proof obligations become simpler thanks to refinement.

Moreover, regarding the approaches developed in the literature, our work covers the whole characteristics of the formal verification of web services compositions. Indeed, the generated Event-B models support the verification of the properties related to both data (interpretation of data) and services (services orchestration). The BPEL2B tool, presented as an Eclipse Plug-in, encodes the interpretation process described in this paper and contributes to the dissemination of formal methods. The details of the formal modelling activities are hidden to the BPEL designer.

This work opens several perspectives. One of them relates to the transactional Web services, mentioned at the end of section 6. Indeed, BPEL offers resources for fault and compensation constructs to handle internal and/or external runtime errors of the described composed service. These constructs are particularly useful for describing transactional services. As a further study we propose to define a complete methodology that extends the one presented in this paper to allow the users to detect and encode transactional parts in web services compositions.

Another perspective relates to the explicit semantics carried by the services. For example, composing in sequence a service that produces distances expressed in centimeters with another one consuming distances expressed in inches should not be a valid composition. Up to now, our approach handles implicit semantics only, it does not handle such a composition. Formal knowledge models carried out by ontologies expressed beside the Event-B models should be investigated.

References

1. van der Aalst, W.M.P., Mooij, A.J., Stahl, C., Wolf, K.: Service Interaction: Patterns, Formalization, and Analysis. In: Bernardo, M., Padovani, L., Zavattaro, G. (eds.) SFM 2009. LNCS, vol. 5569, pp. 42–88. Springer, Heidelberg (2009)
2. Abrial, J.R.: The B-book: assigning programs to meanings. Cambridge University Press, New York (1996)

3. Abrial, J.R.: Modeling in Event-B: System and Software Engineering. Cambridge University Press (2010)
4. Abrial, J.R., Hallerstede, S.: Refinement, Decomposition, and Instantiation of Discrete Models: Application to Event-B. Fundamenta Informaticae 77, 1–28 (2007)
5. Ait-Ameur, Y., Baron, M., Kamel, N., Mota, J.M.: Encoding a process algebra using the Event B method. International Journal on Software Tools for Technology Transfer (STTT) 11(3), 239–253 (2009)
6. Ait-Sadoune, I., Ait-Ameur, Y.: Animating Event B Models by Formal Data Models. In: Margaria, T., Steffen, B. (eds.) ISoLA 2008. CCIS, vol. 17, pp. 37–55. Springer, Heidelberg (2008)
7. Ait-Sadoune, I., Ait-Ameur, Y.: From BPEL to Event-B. In: International Workshop on Integration of Model-based Methods and Tools at IFM Conference (2009)
8. Ait-Sadoune, I., Ait-Ameur, Y.: A Proof Based Approach for Modelling and Verifying Web Services Compositions. In: 14th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS), pp. 1–10. IEEE Computer Society, Potsdam (2009)
9. Aït Sadoune, I., Aït Ameur, Y.: A proof based approach for formal verification of transactional BPEL web services. In: Frappier, M., Glässer, U., Khurshid, S., Laleau, R., Reeves, S. (eds.) ABZ 2010. LNCS, vol. 5977, pp. 405–406. Springer, Heidelberg (2010)
10. Ait-Sadoune, I., Ait-Ameur, Y.: Stepwise Design of BPEL Web Services Compositions, An Event B Refinement Based Approach. In: Lee, R., Ormandjieva, O., Abran, A., Constantinides, C. (eds.) SERA 2010. SCI, vol. 296, pp. 51–68. Springer, Heidelberg (2010)
11. ter Beek, M.H., Bucchiarone, A., Gnesi, S.: Formal methods for service composition. Annals of Mathematics, Computing and Teleinformatics 1(5), 1–10 (2007)
12. Börger, E., Thalheim, B.: Modeling Workflows, Interaction Patterns, Web Services and Business Processes: The ASM-Based Approach. In: Börger, E., Butler, M., Bowen, J.P., Boca, P. (eds.) ABZ 2008. LNCS, vol. 5238, pp. 24–38. Springer, Heidelberg (2008)
13. Brodt, R.: BPEL Designer Project (April 2012), `http://www.eclipse.org/bpel/`
14. Dijkstra, E.W.: A Discipline of Programming, 1st edn. Prentice Hall PTR, Upper Saddle River (1977)
15. Fahland, D., Reisig, W.: ASM-based semantics for BPEL: The negative Control Flow. In: 12th International Workshop on Abstract State Machines, pp. 131–151 (2005)
16. Farahbod, R., Glässer, U., Vajihollahi, M.: An Abstract Machine Architecture for Web Service Based Business Process Management. In: Bussler, C.J., Haller, A. (eds.) BPM 2005 Workshops. LNCS, vol. 3812, pp. 144–157. Springer, Heidelberg (2006)
17. Foster, H.: A Rigorous Approach to Engineering Web Service Compositions. Ph.D. thesis, University of London (2006)
18. Foster, H., Uchitel, S., Magee, J., Kramer, J.: Model-based Verification of Web Service Compositions. In: 18th IEEE International Conference on Automated Software Engineering (ASE 2003), pp. 152–163 (2003)
19. Foster, H., Uchitel, S., Magee, J., Kramer, J.: LTSA-WS: A Tool for Model-Based Verification of Web Service Compositions and Choreography. In: 28th International Conference on Software Engineering, pp. 771–774 (2006)
20. Hinz, S., Schmidt, K., Stahl, C.: Transforming BPEL to petri nets. In: van der Aalst, W.M.P., Benatallah, B., Casati, F., Curbera, F. (eds.) BPM 2005. LNCS, vol. 3649, pp. 220–235. Springer, Heidelberg (2005)

21. Hoare, C.A.R.: An axiomatic basis for computer programming. ACM 12, 576–580 (1969)
22. Lohmann, N.: A Feature-Complete Petri Net Semantics for WS-BPEL 2.0. In: Dumas, M., Heckel, R. (eds.) WS-FM 2007. LNCS, vol. 4937, pp. 77–91. Springer, Heidelberg (2008)
23. Marconi, A.: Automated Process-level Composition of Web Services: from Requirements Specification to Process Run. Ph.D. thesis, University of Trento, Italy (2008)
24. Marconi, A., Pistore, M.: Synthesis and Composition of Web Services. In: Bernardo, M., Padovani, L., Zavattaro, G. (eds.) SFM 2009. LNCS, vol. 5569, pp. 89–157. Springer, Heidelberg (2009)
25. Nakajima, S.: Lightweight formal analysis of Web service flows. Progress in Informatics, 57–76 (2005)
26. Nakajima, S.: Model-Checking Behavioral Specification of BPEL Applications. Electronic Notes in Theoretical Computer Science 151, 89–105 (2006)
27. OASIS: Web Services Business Process Execution Language Version 2.0 (April 2007), http://bpel.xml.org/
28. OMG: Business Process Model and Notation (BPMN) Version 2.0 (June 2010), http://www.omg.org/spec/BPMN/2.0
29. Rodin: User Manual of the RODIN Platform (October 2007), http://deploy-eprints.ecs.soton.ac.uk/11/1/manual-2.3.pdf
30. Salaün, G., Bordeaux, L., Schaerf, M.: Describing and reasoning on web services using process algebra. In: IEEE International Conference on Web Services (ICWS 2004), pp. 43–51 (2004)
31. Salaün, G., Ferrara, A., Chirichiello, A.: Negotiation Among Web Services Using LOTOS/CADP. In: Zhang, L.-J., Jeckle, M. (eds.) ECOWS 2004. LNCS, vol. 3250, pp. 198–212. Springer, Heidelberg (2004)
32. Verbeek, H., van der Aalst, W.M.P: Analyzing BPEL processes using Petri nets. In: 2nd International Workshop on Applications of Petri Nets to Coordination, Workflow and Business Process Management (2005)
33. W3C: OWL-S: Semantic Markup for Web Services (November 2004), http://www.w3.org/Submission/OWL-S/
34. W3C: Web Service Definition Language (WSDL 1.1) (February 2004), http://www.w3.org/TR/wsdl
35. W3C: Web Services Choreography Description Language Version 1.0 (November 2005), http://www.w3.org/TR/ws-cdl-10/
36. WMC-WS: Process Definition Interface - XML Process Definition Language (October 2008), http://www.wfmc.org/xpdl.html

Computing Skyline Incrementally in Response to Online Preference Modification

Tassadit Bouadi[1], Marie-Odile Cordier[1], and René Quiniou[2]

[1] IRISA - University of Rennes 1
[2] IRISA - INRIA Rennes
Campus de Beaulieu, 35042 RENNES, France
{tassadit.bouadi,marie-odile.cordier}@irisa.fr,
rene.quiniou@inria.fr

Abstract. Skyline queries retrieve the most interesting objects from a database with respect to multi-dimensional preferences. Identifying and extracting the relevant data corresponding to multiple criteria provided by users remains a difficult task, especially when the dataset is large. EC^2Sky, our proposal, focuses on how to answer efficiently skyline queries in the presence of dynamic user preferences and despite large volumes of data. In 2008-2009, Wong et al. showed that the skyline associated with any preference on a particular dimension can be computed, without domination tests, from the skyline points associated with first order preferences on that same dimension. Consequently, they propose to materialize skyline points associated with the most preferred values in a specific data structure called *IPO-tree* (*Implicit Preference Order Tree*). However, the size of the *IPO-tree* is exponential with respect to the number of dimensions. While reusing the *merging property* proposed by Wong et al. to deal with the refinements of preferences on a single dimension, we propose an incremental method for calculating the skyline points related to several dimensions associated with dynamic preferences. For this purpose, a materialization of linear size which allows a great flexibility for dimension preference updates is defined. This contribution improves notably the execution time and storage size of queries. Experiments on synthetic data highlight the relevance of EC^2Sky compared to *IPO-Tree*.

1 Introduction

Skyline queries represent a powerful tool for decision-making. Such queries aim at retrieving the most interesting objects from a database with respect to given criteria. In a multidimensional space where the dimension domains are ordered, skyline queries return the points which are not dominated by any other point. A point p dominates a point q if p is strictly better than q on at least one dimension and p is better or equal than q on the remaining dimensions. Identifying and extracting relevant data is often a difficult task especially when dealing with large volumes of data that can be compared according to many criteria. Several studies [15, 20, 22, 24, 18, 11, 10, 4] were carried out on skyline analysis as

A. Hameurlain et al. (Eds.): TLDKS X, LNCS 8220, pp. 34–59, 2013.

a retrieval tool in a decisional context. Skyline queries can formulate multi-criteria queries [16] and obtain the top answers, for example to find the cheapest hotels close to the beach. In the 60s, the search for skyline points was known as the problem of finding admissible points [3] or maximum vectors [2] or Pareto sets. These algorithms have been proved to be ineffective in the case of large databases with many points and many dimensions. Moreover, most of the work mentioned above assume that there exists a predefined order on the domain of each dimension.

An interesting problem arises when users are allowed to define or to change their own preferences online. Thus, on some dimensions the order may change dynamically. This problem is challenging when there are many data points in the dataset and it has attracted the attention of recent work [20, 22]. In fact, the skyline evolves when the preferences change. A naive solution is to recalculate the skyline from scratch for each dynamic preference that has changed. However, it is too expensive on large databases of high dimensionality. The challenge is thus the following: how to efficiently recalculate the least amount of skyline points while minimizing the required memory space.

The solutions proposed by Wong et al. in [22, 20] develop semi-materialization methods to support online query answering for skyline queries involving dynamic preferences. Precisely, the authors of [20] introduced the concept of *n-th order preference*. They showed that the skyline associated with any preference on a particular dimension can be computed from first order preferences on that same dimension. Relying on this *merging property*, they propose to materialize the skyline associated with first order preferences in a specific data structure, called *IPO-tree*, to speed up online query computations. However, to cope with the multiple dimension case they propose to store every possible combination of first-order preferences in an IPO-tree and, so, the size of an IPO-tree is in $O(c^m)$ where m is the number of dimensions with dynamic preferences and c the cardinality of a dimension. In the context of large volume of multidimensional data, this can be intractable. In [22], Wong et al. propose another structure, called *CST* (*Compressed ordered Skyline Tree*), to materialize all possible preference orders. However, this method turns to be incomplete and very complex and, so, it cannot be used. An erratum will be published in [21] for this issue.

The *merging property* of Wong et al. works on one dimension at a time and, thus, is of limited interest. We investigate the case of an arbitrary number of dimensions. Our proposition, EC^2Sky, focuses on how to answer efficiently skyline queries in the presence of several dynamic user preferences despite of large volume of data. This work improves and extends the contribution presented in [5]. Indeed, we extend the related work, introduce three new algorithms related to the EC^2Sky approach, prove the consistency and completeness of the EC^2Sky theorem, and conduct new extensive experiments on large-scale datasets with higher cardinality dimensions and higher dimensional space than the latest experiments presented in [5].

The main idea relies on the incremental addition of dynamic dimensions when computing the skyline. As a side effect, EC^2Sky can return the most relevant

knowledge by emphasizing the compromises associated with the specified preferences. The benefits of this proposition are twofold. On the one hand, complexity in space of the materialization of precomputed skyline reduces to $O(c*m)$ where m is the number of dimensions and c is the size of dimension domains. On the other hand, the number of dominance tests decreases significantly. Some extra memory space and additional runtime is needed to compute skyline related to first-order preferences with respect to Wong et al.'s method. But we proved experimentally that the total computation cost is much lower than in Wong et al.'s method. This contribution enables an incremental computation of skyline points associated with a set of preferences as well as make the interactive modification of preferences easier.

The rest of the paper is organized as follows. In Section 2, we discuss related work on the problem of searching skyline points in the presence of dynamic preferences. In Section 3, we introduce the basic concepts related to skyline queries and dynamic preferences. We develop the formal aspects of our new approach EC^2Sky in Section 4 and present its implementation in Section 5. In Section 6, we present the results of the experimental evaluation performed on synthetic datasets and highlight the relevance of the proposed solution by comparing it to references in the field. We conclude the paper in Section 7.

2 Related Work

Much work has been done on skyline computation. Börzsönyi et al. [4] first investigate and introduce the skyline computation problem in the context of databases, and also propose a block-nested loops (BNL) method and a divide-and-conquer method. Since this pioneering work, many algorithms have been developed for efficient skyline computation. Especially, index-based techniques were proposed in several works to further accelerate skyline queries. The first skyline algorithm incorporated with B-tree or R-tree indexes is proposed in [4]. Then, two progressive processing methods, Bitmap and Index, were proposed in [17]. Skyline queries have been studied in different computational environments, such as full-space skyline retrieval [8, 9, 13, 23] which computes the skyline in a space of a fixed dimensionality, or subspace skyline retrieval [15, 14, 10, 11, 24] where different users may issue skyline queries regarding different subspaces of different dimensions. Skyline queries have also been investigated with various constraints as ranking skyline [18, 6] that enables the user to retrieve the top-K skyline points instead the whole skyline, or metric skyline [7] which retrieves skyline points with dynamic dimensions in the metric space. There have been a considerable number of methods on skyline computation in the context of databases with totally ordered dimensions. However, in real applications, data may include dimensions that are partially ordered in nature, such as categorical (i.e. nominal) dimensions, etc. Some recent studies consider partially ordered dimensions on skyline computation [1], and provide a way to verify dominance among incomparable points over the partial order.

However, most of the work mentioned above assume that there exists only one predefined order on the domain of each dimension. In particular, users cannot express online preferences between dimensions nor customize the preferences between the elements of a given dimension and search the skyline points associated with these preferences. Therefore, other types of skyline queries have been proposed to handle:

- inter-dimension preferences allowing the user to specify the importance of various dimensions and thus to rank the skyline by order of preferences [12]. For example, the price can be considered more important than the distance to the beach. Instead of returning the whole set of skyline points resulting from a query, only the best K points, the top K [6], with respect to the defined inter-dimensions preferences are returned,
- intra-dimension preferences used in [20, 22], allowing each user to express preferences on the different values of a dimension. This kind of preferences is especially attractive for nominal dimensions where there is no evidence of consensual ordering. For example, a user may prefer an hotel of the group *Tulips* to those of the group *Horizon* while another user prefers hotels of the group *Mozilla* to all others.

In this paper, we are particularly interested in skyline queries with dynamic intra-dimension preferences. A naive solution would be to enumerate all possible combinations of preferences and to store the associated skyline. However, the preprocessing burden and the storage induced by a complete materialization of these preferences are prohibitively expensive on large databases. Therefore, other materialization methods have been proposed to support online query answering with dynamic intra-dimensions preferences.

Wong et al. [20] proposed a semi-materialization method based on a specific data structure called IPO-tree (*Implicit Preference Order Tree*). An IPO-tree stores partial useful results corresponding to every combination of first order preferences. A first order preference states that one value is most preferred in some dimension and that the other values are left unordered. An *n-th* order preference specifies an order over n values from some dimension, whereas the other values are less preferred and left unordered. Wong et al. also introduced a property called the *merging property* which makes possible to derive skyline of any n order preference by simple operations on the first order preferences on the same dimension. However, this approach has a main drawback. The *merging property* is applicable to only one dimension at a time. The size of an IPO tree is thus in $O(c^m)$ (where m is the number of dimensions associated with dynamic preferences and c is the cardinality of a dimension). In the context of large databases of high dimensionality this structure becomes very complex.

In [22], Wong et al. proposed another structure, called CST (*Compressed Ordered skyline Tree*), to materialize all the preference orders. They store the skyline with respect to various refinement orders in a compact data structure. However, a CST tree is very complex. This makes updating preferences a difficult task that requires extensive maintenance and changes in the CST tree.

Also, the method is not complete. In fact, the CST tree is constructed gradually by adding dimensions associated with dynamic preferences one by one. During this construction, some points are disqualified from the skyline when adding a new dimension, while they should be in the skyline. An erratum will be published in [21] for this issue.

While reusing the *merging property* proposed by Wong et al. to deal with the refinements of preferences on a single dimension, we propose an incremental method, named EC^{2}Sky, for calculating the skyline points related to several dimensions associated with dynamic preferences. Indeed, the skyline may change due to dimension preference updates, and hence should be incrementally maintained to avoid re-evaluation from scratch. Unlike the IPO-tree method, the EC^{2}Sky structure facilitates and allows a great flexibility for updating dimensions and dynamic preferences.

3 Basic Concepts

In this section, we present the necessary concepts and definitions related to skyline queries. Many are borrowed from [20, 22]. The notations are summarized in Table 1.

The various definitions are illustrated using the example of Table 2 which describes proposals for travels according to the dimensions Price, Distance from the beach, Hotel group (Gr) and Airline (Air).

Table 1. Summary of notations

Notation	Description
E	Dataset
$\|E\|$	Cardinality of E
$D = S \bigcup Z$	Data space of E
S	Subspace with static preferences
Z	Subspace with dynamic preferences
$\|D\|$	Cardinality of D
d_i	One dimension of D ($1 \leq i \leq \|D\|$)
D'	Subspace of D: $D' \subseteq D$
D^i	Subspace of D: $D^i = D^{i-1} \bigcup \{d_i\}$
$\mathcal{P}(E)$	Power set of E
p, q	Data points
$p(d_i)$	Value of p on dimension d_i
$dom(d_i)$	Dimensional domain of d_i
$\wp$	Preferences on Z
$\wp_i$	Preferences on d_i

Table 2. A set of hotels

Hotel ID	Price	Distance	Hotel group	Airline
a	1600	4	T (Tulips)	G(Gonna)
b	2400	1	T(Tulips)	G(Gonna)
c	3000	5	H(Horizon)	G(Gonna)
d	3600	4	H(Horizon)	R(Redish)
e	2300	2	T(Tulips)	R(Redish)
f	3000	3	M(Mozilla)	W(Wings)
g	3600	4	M(Mozilla)	R(Redish)
h	3000	3	M(Mozilla)	R(Redish)

Example 1. *$E = (a, b, c, d, e, f, g, h)$ is a dataset defined in a 4-dimensional space $D =$ (Price, Distance, Hotel group, Airline), $|E| = 8$, $|D| = 4$. The value of point p on dimension Price is denoted by $p(Price) = 1600$.*

Definition 1. ***(Preference order)*** *A preference order on the domain of a dimension d_i is defined by a partial order $\leq_{d_i}$. For two values $p(d_i)$ and $q(d_i)$ in the domain of d_i, we write $p(d_i) \leq_{d_i} q(d_i)$ if the value $p(d_i)$ is preferred to the value $q(d_i)$. We denote $p(d_i) <_{d_i} q(d_i)$ if $p(d_i) \leq_{d_i} q(d_i)$ and $q(d_i) \not\leq_{d_i} p(d_i)$,*

Example 2. *In Table 2, both dimensions, **Price** and **Distance**, are totally ordered by the relation $\leq_{d_i}$, which corresponds to the order relation $\leq$ indicating the smallest of two real numbers. The preference means that the lower the price and the distance, the more preferable a hotel. No order is given a priori on the dimensions Hotel group and Airline. It is up to users to express their own preferences among values belonging to these dimensions.*

Definition 2. ***(Preference type)*** *We distinguish two types of preferences:*

- *static preferences: they correspond to a predefined order relation,*
- *dynamic preferences: they correspond to an order relation that can vary from one user to another or from one user session to another.*

By abuse of language we use *dynamic (resp. static) dimension* instead of *dimension associated with dynamic (resp. static) preferences*. In the rest of this paper, we denote by S the subspace associated with static preferences and by Z the subspace associated with dynamic preferences, with $D = S \bigcup Z$ and $S \bigcap Z = \emptyset$.

Example 3. *$S = \{Price, Distance\}$: the values of these two dimensions follow the order relation $\leq$ specifying that the lower the price (resp. the distance), the more preferable the hotel (ex: $a(Price) <_{Price} d(Price)$). This order is accepted by any user, so it is static. For $Z = \{Gr, Air\}$ no order relation is defined a priori (i.e., in a static way) on these dimensions. The definition of an order is left to users and may vary from one user to another.*

Definition 3. ***(Dominance relation)*** *p dominates q on $D' \subseteq D$, denoted by $p \prec_{D'} q$, if p is preferred or equal to q on any dimension of D' and p is preferred to q on at least one dimension:*
$\forall d_i \in D'$, $p(d_i) \leq_{d_i} q(d_i) \wedge \exists d_i \in D'$, $p(d_i) <_{d_i} q(d_i)$.
$p =_{D'} q$ denotes the fact that p is equally preferred to q on any dimension of D'. When $D' = D$, $p \prec_{D'} q$ is simply noted $p \prec q$.

Example 4. *Let a customer looking for a hotel that is both close to the beach and affordable. In this case hotel a dominates hotel d ($a \prec_{(Price,Distance)} d$) since*
$a(Price) <_{Price} d(Price)$ *and*
$a(Distance) =_{Distance} d(Distance)$.
Hotel a does not dominate hotel b ($a \not\prec_{(Price,Distance)} b$) since
$b(Distance) <_{Distance} a(Distance)$.

The following definitions concern a subspace $D' \subseteq D$. Obviously, these definitions can be generalized to the full dimension space D. The skyline set, or simply the skyline, of a dataset on a subspace contains the points in the dataset that are not dominated by any other point in that dataset.

Definition 4. ***(Skyline)*** *The skyline set of the dataset E on the subspace D' with Z being the subspace associated with the dynamic preferences $\wp$ is the set of points that are not dominated by any point in E:*
$Sky(D', E)_{(Z,\wp)} = \{p \in E |\ \forall q \in E,\ q \not\prec_{D'} p\}$.

If $\wp = \emptyset$, $Sky(D', E)_{(Z,\wp)}$ is simply written $Sky(D', E)$.

Example 5. $Sky(\{Price, Distance\}, E) = \{a, b, e\}$.
$Sky(\{Price, Distance, Gr, Air\}, E) = \{a, b, e, c, d, f, h\}$ *when no preferences are given on dimensions Gr and Air. The points c, d, f and h are no longer dominated by the skyline points $\{a, b, e\}$ since no point can dominate on the unordered dimensions Gr and Air. The point g is dominated by the point h on the static dimensions and have the same values on the dynamic dimensions and so is not in the skyline.*

The set $Sky(D', E)$ contains points, denoted by $MaxSky(D', E)$, that are the best along at least one dimension. It also contains, and this is a major interest of this approach, points denoted $CompSky(D', E)$ that are not dominant on any dimension of D' while being better than any point of E on at least one dimension. These points represent interesting *compromise* solutions for the user from a decision making point of view.

Definition 5. ***(Partition of skyline sets)***
The skyline set of the subspace $D' \subseteq D$ can be decomposed into two sets of points, $MaxSky$ and $CompSky$.
$Sky(D', E) = MaxSky(D', E) \bigcup CompSky(D', E)$ *with :*

- $MaxSky(D', E) = \{p \in Sky(D', E) |\ \exists D'' \subseteq D',\ \forall q \in E,\ p \preceq_{D''} q\}$,
- $CompSky(D', E) = \{p \in Sky(D', E) |\ \forall q \in E,\ q \neq p,\ \exists D'' \subseteq D',\ p \prec_{D''} q\}$.

When D' is restricted to only one dimension ($D' = \{d_i\}$),
$Sky(D', E) = MaxSky(D', E)$.

Example 6. *Let $D' = \{Price, Distance\}$.*
$Sky(\{Price, Distance\}, E) = \{a, b, e\}$.
$MaxSky(\{Price, Distance\}, E) = \{a, b\}$ *since $a(Price)$ (resp. $b(Price)$) is the most preferred value on dimension Price (resp. Distance).*
$CompSky(\{Price, Distance\}, E) = \{e\}$ *since the value of e is not the most preferred either on Price or on Distance. However, $e(Price)$ is preferred to $b(Price)$ on dimension Price and $e(Distance)$ is preferred to $a(Distance)$ on dimension Distance. So, e is better than any $MaxSky$ point on at least one dimension.*

Various kinds of skyline queries can be formulated. Conventional skyline queries retrieve the most interesting objects of a multidimensional dataset. Our goal is to aid a user explore his dataset by letting him express various preferences on dynamic dimensions and assess the consequences of such choices by retrieving the most preferred points i.e. the skyline points. For example, let Hotel group be a dimension with dynamic preferences. Different users may have different preferences on that dimension. If a customer prefers the Hotel group $Horizon$ to all the other hotels, c, d, f and h are added in $Sky(\{Price, Distance, Gr\}, E)$ since they become the best along the Hotel group dimension. However, for another customer preferring $Tulips$ to all the others, c, d, f and h do not belong to the skyline since they are dominated by a, b and e. An interesting observation is that hotels associated with a, b and e are always in the skyline no matter which preference order on the Hotel group is chosen (because a is the only one with the best price, b is the only one with the best distance from the beach and e represent a compromise for dimensions Price and Distance).

When a user formulates a query involving a dimension d_i with dynamic preferences, she/he can specify the preference order on the $|d_i|$ values of this dimension. The order is total if all these values are ordered. But this is not always possible and the user may order only n of the $|d_i|$ values. Implicitly, she/he considers that they are more preferred than the $(|d_i| - n)$ remaining values which are left unordered. This corresponds to the notion of *n-th order implicit* preference introduced in [20].

Definition 6. ***(n-th order preference)*** *Let $d_i \in Z$ and $|d_i| = m$. $\wp_i$ is an n-th order preference on d_i iff :*

- $\wp_i = v_1 <_{d_i} \ldots <_{d_i} v_n <_{d_i} *$, *with* $v_1 \in dom(d_i), \ldots, v_n \in dom(d_i)$ *and* $n \leq m$,
- $\forall k \in \{n+1, .., m\}, v_n <_{d_i} v_k$.

When $n = 1$, $\wp_i = v_1 <_{d_i} *$ is called a first order preference.

Thus $<_{d_i}$ is a total order on the values $\{v_1 \ldots v_n\}$ of d_i, and a partial order on the whole dimensional domain of d_i.
Note the importance of first order preferences: they are sufficient to determinate the dominant points of a dimension.

Example 7. *For the dimension Hotel group in Table 2, a user prefers* $T(Tulips)$ *to* $M(Mozilla)$, T *to* $H(Horizon)$ *and* M *to* H *(i.e.,* $T <_{Gr} M <_{Gr} H$*). This preference is a third order preference and defines a total order. Some other user could prefer Hotel group* T *to any other group (i.e.,* $T <_{Gr} *$*). In this case, the preference is a first order preference which defines the partial order* $\{T <_{Gr} M, T <_{Gr} H\}$.

$\wp_i = v_1 <_{d_i} \ldots <_{d_i} v_n <_{d_i} *$ denotes the set of binary preferences $\wp_i = \{v_1 <_{d_i} v_2, v_2 <_{d_i} v_3, \ldots, v_n <_{d_i} *\}$. The absence of preference on dimension d_i is denoted by $\wp_i = \emptyset$. In the sequel, we use both notations for $\wp_i$.

Example 8. *Let* $Z = \{Gr\}$, $\wp = \{H <_{Gr} *\}$ *(equivalent to* $\{H <_{Gr} M, H <_{Gr} T\}$*).*
Then $Sky(D,E)_{(Z,H<_{Gr}*)} = \{a,b,e,c,d,f,h\}$*. The points c and d are no longer dominated by the skyline points* $\{a,b,e\}$ *since they have the best values on the dimension Gr. The values M and H of the dimension Gr are left unordered, so the points f and h become compromise skyline points because they are no longer dominated by the skyline points* $\{a,b,e\}$ *on the dimension Gr.*

We give below some useful properties of the preference relationship. These properties will be used later to reduce the number of domination tests during skyline computation. In the following, $\wp = \bigcup_{i=1}^{|Z|} \wp_i$ denotes the set of dynamic preferences associated with $Z \subseteq D$ combined implicitly with the set of static preferences associated with $S \subseteq D$.

Definition 7. ***(Preference inclusion)*** *Let* $|Z| = m$, $\wp = \{\wp_1, .., \wp_m\}$ *and* $\wp' = \{\wp'_1, .., \wp'_m\}$*, where every* $\wp_i$ *and* $\wp'_i$ *are sets of binary preferences associated with the dimension* $d_i \in Z$.
Then, $\wp \subseteq \wp'$ *if and only if* $\wp_i \subseteq \wp'_i$ *for* $1 \leq i \leq m$.

Definition 8. ***(Preference refinement)*** *Let* $\wp'$ *and* $\wp''$ *two preferences sets on the subspace* Z. $\wp''$ *is a refinement of* $\wp'$ *if* $\wp' \subseteq \wp''$.

Property 1. ***(Monotonicity of preference refinement)*** *Let* $\wp'$ *and* $\wp''$ *two preferences sets on* Z*. If* $\wp''$ *is a refinement of* $\wp'$ *then* $Sky(D,E)_{(Z,\wp'')} \subseteq Sky(D,E)_{(Z,\wp')}$.

The following example illustrates property 1.

Example 9. *Let* $Z = \{Gr\}$,
$\wp' = \{H <_{Gr} *\}$ *and* $\wp'' = \{H <_{Gr} T <_{Gr} *\}$.
$\wp''$ *is a refinement of* $\wp'$ *since* $\wp' \subset \wp''$.
$Sky(D,E)_{(Z,\wp')} = \{a,b,c,d,e,f,h\}$ *and* $Sky(D,E)_{(Z,\wp'')} = \{a,b,c,d,e\}$.
We have $Sky(D,E)_{(Z,\wp'')} \subset Sky(D,E)_{(Z,\wp')}$.

Property 1 indicates that when preferences are refined, the skyline may become smaller and so, some skyline points may be disqualified. Also, if a point is not in the skyline related to some preference, it won't belong to the skyline

related to a refined preference. We will use property 1 later to reduce the number of domination tests in our approach.

The following theorem formulates an important property called the *merging property* that was introduced by Wong et al. [20]. This property provides a means to derive the skyline related to any possible n-th order preference by operations on first order preferences on the same dimension.

Theorem 1. ***(Merging property)*** *Let $\wp^{'}$ and $\wp^{''}$ be two preferences differing only on dimension d_i, i.e. $\wp^{'}_j = \wp^{''}_j$ for all $j \neq i$. Let $\wp^{'}_i = v_1 <_{d_i} \ldots < v_{k-1} <_{d_i} *$ and $\wp^{''}_i = v_k <_{d_i} *$. Let $PSky(D,E)_{(Z,\wp')}$ be the set of points in $Sky(D,E)_{(Z,\wp')}$ with d_i values in $\{v_1 \ldots v_{k-1}\}$. Let $\wp^{'''}$ be a preference differing from $\wp^{'}$ and $\wp^{''}$ only on dimension d_i and $\wp^{'''}_i = v_1 < \ldots < v_{k-1} < v_k < *$. Then the skyline associated with $\wp^{'''}$ is:*

$$Sky(D,E)_{(Z,\wp''')} = (Sky(D,E)_{(Z,\wp')} \bigcap Sky(D,E)_{(Z,\wp'')}) \bigcup PSky(D,E)_{(Z,\wp')}.$$

Example 10. *Let* $\wp^{'} = \{M <_{Gr} *\}$, $\wp^{''} = \{H <_{Gr} *\}$,
$\wp^{'''} = \{M <_{Gr} H <_{Gr} *\}$ *and* $Z = \{Group\}$.
$Sky(D,E)_{(Z,\wp''')} = (Sky(D,E)_{(Z,\wp')} \bigcap Sky(D,E)_{(Z,\wp'')}) \bigcup PSky(D,E)_{(Z,\wp')}$
$= (\{a,b,e,f,h\} \bigcap \{a,b,e,c,d,f,h\}) \bigcup \{f,h\}$
$= \{a,b,e,f,h\}$

Wong et al. have proposed successively two interesting methods, IPO-tree [20] and CST [22], for skyline computation based on the properties and theorem 1 presented above. However, the implementation based on these two proposals raises several problems:

- The size of the IPO-Tree structure is in $O(c^m)$ where m is the number of dimensions with dynamic preferences and c the cardinality of a dimension. So it is intractable in the context of large databases with high dimensionality and does not allow scaling. It is worth-noting that the *merging property* is applicable to only one dimension at a time,
- The CST method does not solve the *IPO-Tree* problems because its algorithm is incomplete (it disqualifies points which should be in the skyline).

The second proposal (CST) is incomplete [21], thus we focus on the first proposal (IPO-Tree). The IPO-Tree method supports the refinement of preferences. However, it addresses the treatment of one dynamic dimension at a time (merging property). To cope with several dynamic dimensions, Wong et al. propose to store every combination of the first order preferences related to these dimensions. So, the size of the proposed materialization structure is exponential, which is prohibitive when dealing with several dimensions. We propose in Section 4 an incremental method which makes possible to introduce dynamic dimensions one by one. It relies on a structure that enables an effective materialization of dynamic preferences.

4 EC²Sky: An Incremental Skyline Computation

In this section we introduce the proposed incremental method and the theorem that grounds the method.

Let us examine how the addition of a dynamic dimension d_i impacts the skyline that was previously computed for the dimension subspace D^{i-1}. Intuitively, the computation of the new skyline for $D^i = D^{i-1} \bigcup d_i$ is a two-step process. First, compute the skyline associated to the new dynamic dimension union the static dimensions $d_i \bigcup S$, as if it were independent of the other dynamic dimensions. Second, take into account the correlations between the new dimension d_i and the previous dimensions D^{i-1} to update the new computed skyline. This second task consists in, i) removing from the skyline independently computed for $d_i \bigcup S$ the points that are disqualified i.e. are dominated on the dynamic dimensions of D^{i-1}, ii) removing the set of old skyline points that are disqualified i.e. are dominated on the new dimension d_i, iii) completing the resulting skyline with points that are new compromises for $D^{i-1} \bigcup d_i$.

In the following, we assume that the subset of dimensions D^i is such that $D^i = D^{i-1} \bigcup d_i$, with $d_i \in Z$, $i \in \{1, .., |Z|\}$, $D^i \subseteq D$ and $D^0 = S$. This notation represents the incremental addition of dimensions in skyline computation.

Consider the addition of a dynamic dimension d_i to a set D^{i-1} of $i-1$ dynamic dimensions. As sketched above, the first task is to compute $Sky(d_i \bigcup S, E)_{(Z,\wp)}$, the skyline related to dimension d_i as if it were independent of the other dynamic dimensions. Wong et al.'s method can be used to achieve this task. However, this set may contain skyline points that are disqualified i.e. they are dominated on the dynamic dimensions of the subspace D^{i-1}. Precisely, let $p, q \in Sky(d_i \bigcup S, E)_{(Z,\wp)}$ be two skyline points with the same values on every dimension of $d_i \bigcup S$. If q is preferred on D^{i-1} it will dominate p and disqualify it from the skyline $Sky(D^i, E)_{(Z,\wp)}$. This set of points is denoted $CutSky(d_i \bigcup S, E)$.

Definition 9. ***(Disqualified skyline points from*** $d_i \bigcup S$***)*** *The set of skyline points related to the subspace* $d_i \bigcup S$ *that are disqualified by the introduction of the subspace* D^{i-1} *is defined by* $CutSky(d_i \bigcup S, E) = \{p \in Sky(d_i \bigcup S, E)_{(Z,\wp)} \mid \exists q \in Sky(d_i \bigcup S, E)_{(Z,\wp)},\ p =_{d_i \bigcup S} q \wedge q \prec_{D^{i-1}} p\}$.

Example 11. *Let* $D^{i-1} = D^1 = \{Price, Distance, Air\}$, $d_i = Gr$, *the new dimension, and the preferences:*
$\{M <_{Gr} H <_{Gr} T\}$ *and* $\{W <_{Air} *\}$.
$Sky(D^1, E)_{(Z,\wp)} = \{a, b, e, f\}$ *and* $Sky(\{Gr\} \bigcup S, E)_{(Z,\wp)} = \{a, b, e, f, h\}$.
$CutSky(\{Gr\} \bigcup S, E) = \{h\}$ *as the point h should be removed from the skyline* $Sky(D^2, E)_{(Z,\wp)}$ *since it is dominated by the point f on the subspace* D^1.

On the other hand, the old skyline $Sky(D^{i-1}, E)_{(Z,\wp)}$may contain points that are disqualified by dominant points brought by the new dimension d_i. Precisely, let $p, q \in Sky(D^{i-1}, E)_{(Z,\wp)}$ be two skyline points with the same values on every

dimension of D^{i-1}. If q is preferred on the new dimension d_i, it will dominate p and disqualify it from the skyline $Sky(D^i, E)_{(Z,\wp)}$. This set of points is denoted $CutSky(D^{i-1}, E)$.

Definition 10. ***(Disqualified skyline points from*** D^{i-1}***)*** *The set of skyline points related to the subspace* D^{i-1} *that are disqualified by the introduction of dimension* d_i *is defined by*
$CutSky(D^{i-1}, E) =$
$\{p \in Sky(D^{i-1}, E)_{(Z,\wp)} \mid \exists q \in Sky(D^{i-1}, E)_{(Z,\wp)}, p =_{D^{i-1}} q \wedge q \prec_{d_i \bigcup S} p\}$.

Example 12. *Let* $D^{i-1} = D^1 = \{Price, Distance, Gr\}$, $d_i = Air$, *the new dimension, and the preferences* $\{M <_{Gr} H <_{Gr} T\}$ *and* $\{G <_{Air} R <_{Air} W\}$.
$Sky(D^1, E)_{(Z,\wp)} = \{a, b, e, f, h\}$, $CutSky(\{Air\} \bigcup S, E) = \{\}$ *and*
$CutSky(D^1, E) = \{f\}$ *as the point* f *should be removed from the skyline*
$Sky(D^2, E)_{(Z,\wp)}$ *since it is dominated by the point* h *on the new dimension* $d_i = Air$.

Finally, some new points should appear in the new skyline. Precisely, before taking into account the new dimension d_i, some points may be dominated on every dimension of $D^{i-1} \bigcup S$ and, so, are not in the skyline. But, when dimension d_i is introduced, being better on d_i than some skyline points they were dominated by, they may well be no longer dominated by any skyline point from $Sky(D^{i-1}, E)_{(Z,\wp)}$ on some dimensions from D^{i-1}: they are new compromise skyline points. This set of points is denoted $NewCompSky(D^i, E)$.

Definition 11. ***(New compromise skyline)***
Let $C = Sky(D^{i-1}, E)_{(Z,\wp)} \bigcup Sky(d_i \bigcup S, E)_{(Z,\wp)}$.
The set of new compromise skyline points is defined by
$NewCompSky(D^i, E) = \{p \in E - C \mid \forall q \in E, \exists d_k \in D^i, p \prec_{d_k} q\}$.

Example 13. *Let* $D^{i-1} = D^1 = \{Price, Distance, Gr\}$, $d_i = Air$, *the new dimension, and the preferences:*
$\{M <_{Gr} H <_{Gr} T\}$ *and* $\{G <_{Air} R <_{Air} W\}$.
$Sky(D^1, E)_{(Z,\wp)} = \{a, b, e, f, h\}$, $Sky(\{Air\} \bigcup S, E)_{(Z,\wp)} = \{a, b, e\}$,
$CutSky(\{Air\} \bigcup S, E) = \{\}$ *and* $CutSky(D^1, E) = \{f\}$. *But, if we consider simultaneously the two dimensions* Gr *and* Air *then* c *is no longer dominated by* f. *As* f *was the only point* c *was dominated by,* c *becomes a new skyline point. Since* c *is the only such "promoted" point,* $NewCompSky(D^2, E) = \{c\}$.

Suppose we want to extend a dimensional subspace D^{i-1} with a new dimension d_i. The following theorem states that the skyline of the extended subspace can be computed by removing disqualified skyline points from the old skyline and by adding the new skyline points brought by the preference on the new dimension. The new skyline points are either dominant points on the new dimension or new compromise skyline points introduced by the new preference.

Theorem 2. ***(Incremental skyline)***
Let E *be a* $|D|$*-dimensional dataset,* $Z \subseteq D$ *the subspace of size* $|Z| = m$ *with*

dynamic preferences $\wp = \{\wp_j\}_{j=1,..,m}$ *on* D, $Sky(d_i \bigcup S, E)$ *the skyline of the subspace* $S \bigcup d_i$ *and* $D^i = D^{i-1} \bigcup d_i$, *with* $i = \{1, .., m\}$.

$Sky(D^i, E)_{(Z,\wp)} = (Sky(D^{i-1}, E)_{(Z,\wp)} \bigcup Sky(d_i \bigcup S, E)_{(Z,\wp)}) - (CutSky(D^{i-1}) \bigcup CutSky(d_i \bigcup S)) \bigcup NewCompSky(D^i, E)$.

Example 14. ***(Illustration of Theorem 2)*** *Let* $D^1 = \{Price, Distance, Gr\}$, $D^2 = \{Price, Distance, Gr, Air\}$ *and we consider the preferences of the previous example.*
$Sky(D^2, E)_{(Z,\wp)} =$
$(Sky(D^1, E)_{(Gr, H<_{Gr}M<_{Gr}T)} \bigcup Sky(\{Air\} \bigcup S, E)_{(Air, G<_{Air}W<_{Air}R)})$
$-((CutSky(D^1, E) \bigcup CutSky(\{Air\} \bigcup S, E)) \bigcup NewCompSky(D^2, E) =$
$(\{a, b, e, f, h\} \bigcup \{a, b, e\})$ - $(\{f\} \bigcup \{\}) \bigcup \{c\} =$
$\{a, b, c, e, h\}$.

Proof. **(Theorem 2)** Here follows a sketch of the proof of theorem 2. We consider successively the points that are disqualified from the skyline and the points that are added to the skyline. Let $D^i \subseteq D$ and $D^i = D^{i-1} \bigcup \{d_i\}$. Let Z be the subspace of D with dynamic preferences and $Sky(D^i, E)$ be the skyline of the subspace D^i.

- Any element of $CutSky$ must be disqualified from the resulting skyline.
  If $p \in (CutSky(D^{i-1}) \bigcup CutSky(d_i \bigcup S))$ then there exists some $q \in Sky(D^i, E)$ such that
  $(q =_{D^{i-1}} p$ and $q \prec_{d_i \bigcup S} p)$ or $(q =_{d_i \bigcup S} p$ and $q \prec_{D^{i-1}} p)$.
  This means that $q \prec_{D^i} p$. Thus, p should not be in $Sky(D^i, E)_{(Z,\wp)}$.
- Any element of $Sky(D^i, E)_{(Z,\wp)}$ must belong to
  $\{ (Sky(D^{i-1}, E)_{(Z,\wp)} \bigcup Sky(d_i \bigcup S, E)_{(Z,\wp)}) - (CutSky(D^{i-1}) \bigcup CutSky(d_i \bigcup S)) \bigcup NewCompSky(D^i, E)\}$.
  Any $p \in Sky(D^i, E)_{(Z,\wp)}$ is such that:
  - either there exists a dimension $d_j \in D^i$ such that $\forall q \in E$, $p \preceq_{d_j} q$. In this case, $p \in (Sky(D^{i-1}, E)_{(Z,\wp)} \bigcup Sky(d_i \bigcup S, E)_{(Z,\wp)})$
  - or for every $q \in E$, there exists a $d_i \in Z$ such that $p \prec_{d_i} q$. In this case, $p \in NewCompSky(D^i, E)$ and $p \notin (CutSky(D^{i-1}) \bigcup CutSky(d_i \bigcup S)$

  In both cases p belongs to $Sky(D^i, E)_{(Z,\wp)}$ □

Theorem 2 provides a scheme for an incremental computation of skyline queries associated with several dynamic dimensions. In the following, we describe more precisely our proposal.

5 EC²Sky Implementation

In this section, we present the implementation of incremental skyline computation. We introduce some definitions to characterize the points that are involved

in the incremental computation of skyline points and facilitate the specification of the algorithms and of the materialization structure. To ensure an efficient and online computation of skyline, we provide an effective materialization structure detailed in the sequel. We propose a trade-off between (i) *materialize* all the skyline points for all possible preferences and (ii) *calculate*, for each user query, the skyline points associated with the preferences formulated in the query. Our approach is based on three steps:

1. compute and store the skyline on static dimensions. One can adopt any existing algorithm (e.g. [1]) that computes the skyline for partially ordered domains;
2. for each dimension with dynamic preferences, compute and store the candidate skyline points according to any possible first order preference;
3. rely on the information stored in step 1 and 2 to compute the skyline points related to user preferences on incrementally introduced dynamic dimensions.

5.1 Skyline Associated with Static Dimensions

In step 1 we compute all the skyline points corresponding to the defined static preferences of D. Two concepts introduced by Wong et al. in [22] are helpful. They decompose the set $Sky(D, E)$, corresponding to the defined static preferences of D and denoted by $\wp_\emptyset$, into two subsets: the *global skyline set GSky(D,E)* and the *order-sensitive skyline set OsSky(D,E)*.

The points in the global skyline set $GSky(D, E)$ remain in the skyline whenever any preference on any dimension of Z is added.

Definition 12. ***(Global skyline points)*** *The global skyline set of the space $D = S \bigcup Z$ on the dataset E, is defined by*
$GSky(D, E) =$
$\{p \in Sky(D, E) \mid \forall\ q \in Sky(D, E), \nexists\ d_i \in Z,\ p =_S q \wedge p(d_i) \neq_{d_i} q(d_i)\}$

Some skyline points are qualified order-sensitive because, depending on the preferences associated with dynamic dimensions, these points may be skyline or not. Note first that no global skyline points is order sensitive. Second, $CutSky$ points have to be searched among order sensitive skyline points.

Definition 13. ***(Order-sensitive skyline points)*** *The order-sensitive skyline set of the space D on the dataset E, is defined by*
$OsSky(D, E) = \{p \in Sky(D, E) \mid p \notin GSky(D, E)\}$ *or equivalently*
$OsSky(D, E) = Sky(D, E) - GSky(D, E)$.

Example 15. *Let $S = \{Price,\ Distance\}$ and $Z = \{Gr,\ Air\}$.*
Then $GSky(D, E) = \{a, b, e\}$ and $OsSky(D, E) = \{c, d, f, h\}$, because all the skyline points are distinct.

5.2 Skyline Associated with Dynamic Dimensions

This section details step 2 of our approach. In this step, we pre-compute the useful information that does not depend on the dynamic preferences provided

by users. For each dimension d_i with dynamic preferences, we introduce *the candidate skyline point* (CP_{d_i}), *the new skyline point set* ($NewSky_{(d_i,\wp_i^j)}$) and *the compromise candidate point set* ($CandComp_{(d_i,\wp_i^j)}$).
The set CP_{d_i} represents the points that may become skyline points over the dimension d_i. It is the set of points from $OsSky(D,E)$

(1) having on $d_i \in Z$ a value different from any point of $GSky(D,E)$ that dominates them,
(2) having the same value on the static dimensions but different values on $d_i \in Z$.

In the sequel, $p \prec^j_{d_i \bigcup S} q$ indicates that p dominates q on the subspace $d_i \bigcup S$ according to the first order preference $\wp_i^j$ of the dimension d_i.

Definition 14. ***(Candidate skyline points)*** *The candidate skyline point set of the dynamic dimension d_i, is defined by*
$CP_{d_i} =$
$\{p \in OsSky(D,E) \mid \exists\ q \in GSky(D,E),\ q \prec_S p,\ p(d_i) \neq_{d_i} q(d_i)\} \bigcup$
$\{p \in OsSky(D,E) \mid \exists\ q \in OsSky(D,E),\ q =_S p,\ p(d_i) \neq_{d_i} q(d_i)\}$

Example 16. *Let $S = \{Price, Distance\}$ and $d_i = \{Gr\}$. Then $CP_{Gr} = \{c,d,f,h\}$.*

To find the new skyline after the introduction of the new dimension d_i, it is sufficient to test the points in CP_{d_i} instead of all non-skyline points. This can significantly reduce the number of domination tests.

$NewSky_{(d_i,\wp_i^j)}$ (Algorithm 1) represents the set of points in CP_{d_i} that are preferred to the points in $GSky(D,E)$ according to the first order preference $\wp_i^j = v_j <_{d_i} *$ such that $v_j \in dom(d_i)$. Intuitively, NewSky points are equivalent to MaxSky points on d_i according to the preference $\wp_i^j$

Definition 15. ***(New skyline points)***
The new skyline point set of the dynamic dimension d_i, is defined by
$NewSky_{(d_i,\wp_i^j)} = \{p \in CP_{d_i} \mid \forall\ q \in GSky(D,E) \bigcup \{\ CP_{d_i} - p\},\ q \not\prec^j_{d_i \bigcup S} p\}$

Example 17. $NewSky_{(Gr,H<_{Gr}*)} = \{c,d,f,h\}$.

Finally, $CandComp_{(d_i,\wp_i^j)}$ (Algorithm 2) represents the set of points that may become skyline compromises (i.e. compromise candidates) when considering a new dimension. They are computed for each first order preference $\wp_i^j$ on d_i.

Definition 16. ***(Compromise candidate points)***
Let $E' = (CP_{d_i}$ - $NewSky_{(d_i,\wp_i^j)})$ and $E'' = (GSky(D,E) \bigcup NewSky_{(d_i,\wp_i^j)})$. The compromise candidate points associated with the preference $\wp_i^j$ is a set of pairs (p, Set_p) defined by
$CandComp_{(d_i,\wp_i^j)} =$
$\{(p, Set_p) \in E' \times \mathcal{P}(E'') \mid \forall q \in \mathcal{P}(E''),\ \exists d_k \in \{d_i\} \bigcup S,\ p \prec^j_{d_k} q\}$.

Algorithm 1. Calculate $NewSky_{(d_i,\wp_i^j)}$

input : d_i: a dimension, $\wp_i^j$: a first order preference on d_i, $GSky(D,E)$: global skyline set, CP_{d_i}: candidate skyline set over d_i
output: $NewSky_{(d_i,\wp_i^j)}$

```
1 NewSky_(d_i,℘_i^j) ← ∅
2 foreach p ∈ CP_{d_i} do
3     bool ← true
4     foreach q ∈ GSky(D,E) ⋃ { CP_{d_i} − p} do
5         if q ≺^j_{d_i ⋃ S} p then
6             bool← false
7             exit
8     if bool then
9         NewSky_(d_i,℘_i^j) ← NewSky_(d_i,℘_i^j) ⋃{p}
```

Example 18
$CandComp_{(Air,R<_{Air}*)} = \{(f,\{a,b\})\}$ where the notation $\{(f,\{a,b\})\}$ means that f belongs to $CandComp_{(Air,R<_{Air}*)}$ because f dominates a (resp. b) on at least one dimension from $\{Air\} \bigcup S$ (here Distance (resp. Airline))).

5.3 The EC²Sky Structure

Now, let us consider how to construct an EC^2Sky data structure to store efficiently all the precomputed information. Our aim is to avoid building a data structure containing all the combinations of the dynamic preferences on all dimensions as proposed in [20] . In section 5.1 and 5.2 and thanks to theorem 2, we have shown that the skyline of an extended dimensional subspace can be computed by taking into account first order preferences only. We propose to store in EC^2Sky structure all the sets $NewSky$ and $CandComp$ associated to any first order preference in each dimension.

For each dimension d_i, we compute and store CP_{d_i} and for each first order preference on d_i, we compute and store the two sets: $NewSky_{(d_i,\wp_i^j)}$ and $CandComp_{(d_i,\wp_i^j)}$ related to the first order preference j on dimension d_i. The sets $NewSky_{(d_i,\wp_i^j)}$ and $CandComp_{(d_i,\wp_i^j)}$ associated with any possible first order preference on dimension *Hotel group* or dimension *Airline* are presented in Table 3.

Now, we evaluate the space complexity of the EC^2Sky structure. Let m be the number of dimensions associated with dynamic preferences and c be the maximal cardinality of a dimension associated with dynamic preferences. The space complexity of the EC^{2}Sky structure is given by:

Algorithm 2. Calculate $CandComp_{(d_i,\wp_i^j)}$

input : d_i: a dimension, $\wp_i^j$: a first order preference on d_i, $NewSky_{(d_i,\wp_i^j)}$: skyline points added by d_i for $\wp_i^j$, $GSky(D,E)$: global skyline set, CP_{d_i}: candidate skyline set
output: $CandComp_{(d_i,\wp_i^j)}$

```
1 CandComp_(d_i,℘_i^j) ← ∅
2 foreach p ∈ {CP_di − NewSky_(d_i,℘_i^j)} do
3     Set_p ← ∅
4     foreach q ∈ {GSky(D,E) ⋃ NewSky_(d_i,℘_i^j)} do
5         foreach d_k ∈ {d_i} ⋃ S do
6             if p ≺^j_dk q then
7                 Set_p ← Set_p ⋃ q
8         CandComp_(d_i,℘_i^j) ← CandComp_(d_i,℘_i^j) ⋃ {(p, Set_p)}
```

Table 3. Illustration of an EC^2Sky structure with two dynamic dimensions and three first order preferences for each dimension

$GSky = \{a, b, e\}$					
$CP_{Gr} = \{c, d, f, h\}$			$CP_{Air} = \{f\}$		
$\wp = M <_{Gr} *$	$\wp = T <_{Gr} *$	$\wp = H <_{Gr} *$	$\wp = R <_{Air} *$	$\wp = G <_{Air} *$	$\wp = W <_{Air} *$
$NewSky_{\{Gr,\wp\}}$			$NewSky_{\{Air,\wp\}}$		
$\{f, h\}$	$\{\}$	$\{c, d, f, h\}$	$\{\}$	$\{\}$	$\{f\}$
$CandComp_{\{Gr,\wp\}}$			$CandComp_{\{Air,\wp\}}$		
$\{(c, \{a, b, e\}), (d, \{a, b, e\})\}$	$\{(f, \{a\}), (h, \{a\})\}$	$\{\}$	$\{(f, \{a, b\})\}$	$\{(f, \{a, e\})\}$	$\{\}$

$$\sum_{i=0}^{m}(c) = O(c.m)$$

We can note that the size of the EC^{2}Sky structure is significantly smaller than the number of possible *n-th* order preferences given by:

$$(\sum_{i=0}^{c-1}(P_i(c)))^m = O((c.c!)^m)$$

Where $P_i(c)$ is the number of permutations of ordering i elements from c elements. The space complexity of the EC^{2}Sky structure is also significantly smaller than the space complexity of IPO-tree structure given by:

$$\sum_{i=0}^{m}(c+1)^i = O(c^m)$$

Algorithm 3. Calculate $CutSky(D^{i-1})$

input : $Sky(D^{i-1}, E)$, $Sky(D^i, E)$ and $Sky(d_i \bigcup S, E)$
output: $CutSky(D^{i-1})$

```
1 CutSky(D^{i-1}) ← ∅
2 foreach p ∈ Sky(D^i, E) do
3     foreach q ∈ Sky(D^{i-1}, E) ⋃ Sky(d_i ⋃ S, E) do
4         if p =_{D^{i-1}} q and q ≺_{d_i ⋃ S} p then
5             CutSky(D^{i-1}) ← CutSky(D^{i-1}) ⋃ {p}
6             break
```

For example, when $m = 3$ and $c = 40$, the number of stored preferences in EC^{2}Sky structure is 123 only, while in IPO-tree structure is $70,644$, and the number of all possible *n-th* order preferences ($n \in 1, .., c$) is $4.1 * 10^9$. This is 574.35 times smaller than the IPO-tree and $714,502,572$ times smaller than the number of all possible *n-th* order preferences. The difference is more obvious when the number of dimensions m is high.

5.4 Query Evaluation

In this section, we describe step 3 of our proposal (cf. beginning of section 5). The information precomputed and stored in step 1 and 2 is used in step 3 to calculate, interactively, the skyline set according to the specified preferences in the user query.

One dimension with dynamic preferences First, we consider only one dimension with dynamic preferences in the dimensional space D. According to the user query, we are faced with two cases:
(i) *Query with first order preferences*: to compute the skyline associated with a first-order preference $\wp_i^j$, we use the two sets $GSky(D, E)$ and $NewSky_{(d_i, \wp_i^j)}$ stored in step 2, as follows : $Sky(d_i \bigcup S, E)_{(Z, \wp_i^j)} = GSky(D, E) \bigcup NewSky_{(d_i, \wp_i^j)}$. Recall that, when dealing with one dimension only, there is no compromise points ($CompSky = \emptyset$).

Example 19. *We use the EC^{2}Sky structure in Table 3 to illustrate the different steps of a query evaluation. The skyline associated with the preference $\wp = \{M <_{Gr} *\}$ (stored in the EC^{2}Sky structure shown in Table 3) is computed as follow:*
$Sky(\{Gr\} \bigcup S, E)_{(Z, M <_{Gr} *)} = GSky(D, E) \bigcup NewSky_{\{Gr, M <_{Gr} *\}}$.
Thus, $Sky(\{Gr\} \bigcup S, E)_{(Z, M <_{Gr} *)} = \{a, b, e, f, h\}$, which is the skyline for $\wp$.

Algorithm 4. Calculate $NewCompSky(D^i, E)$

input : EC^2Sky structure, $GSky(D, E)$: global skyline points
output: $NewCompSky(D^i, E)$

1 $CandCompSet = \bigcup^{i} \bigcup^{j} CandComp_{(d_i, \wp_i^j)}$
2 $NewCompSky(D^i, E) \leftarrow \emptyset$
3 **foreach** $p \in CandCompSet$ **do**
  `// Compute the set of points dominated by p on at least one dimension of` D^i
4 $Dominated(p) \leftarrow \{q | \exists\ d_i \in D^i, p \prec_{d_i} q\}$
  `// Select the set of points that dominate the skyline points on at least one dimension`
5 **if** $Dominated(p) = Sky(D^{i-1}, E) \bigcup Sky(d_i \bigcup S, E) - (CutSky(D^{i-1}) \bigcup CutSky(d_i \bigcup S)$ **then**
6 $NewCompSky(D^i, E) \leftarrow NewCompSky(D^i, E) \bigcup \{p\}$
7 Dominance test over all the elements of the set $NewCompSky(D^i, E)$

(ii) *Query with n-th order preferences*: in this case we use the *merging property* of Wong et al. [20] (see Theorem 1). This is illustrated by the following example.

Example 20. *The skyline associated with the preference* $\wp = \{M <_{Gr} H <_{Gr} *\}$ *can be computed from the skyline related to the preferences* $\wp^1 = \{M <_{Gr} *\}$ *and* $\wp^2 = \{H <_{Gr} *\}$ *(stored in the* EC^2Sky *structure shown in Table 3), as follow:*
$Sky(\{Gr\} \bigcup S, E)_{(Z, M <_{Gr} *)} = \{a, b, e, f, h\}$ *(cf. example 19), which is the skyline for* $\wp^1$.
In the same way, $Sky(\{Gr\} \bigcup S, E)_{(Z, H <_{Gr} *)} = \{a, b, e, c, d, f, h\}$, *which is the skyline for* $\wp^2$.

Finally, to compute $\wp = \{M <_{Gr} H <_{Gr} *\}$, *we use the merging property (Theorem 1) :*
$Sky(D, E)_{(Z, M <_{Gr} H <_{Gr} *)} =$
$(Sky(\{Gr\} \bigcup S, E)_{(Z, M <_{Gr} *)} \bigcap Sky(\{Gr\} \bigcup S, E)_{(Z, H <_{Gr} *)}) \bigcup$
$PSky(D, E)_{(Z, M <_{Gr} *)} =$
$(\{a, b, e, f, h\} \bigcap \{a, b, e, c, d, f, h\}) \bigcup \{f, h\} = \{a, b, e, f, h\}$

Several dimensions with dynamic preferences Second, we consider the case of several dimensions with dynamic preferences which is more complex. According to definition 10 and 11, some skyline points ($CutSky$ points) may be disqualified

Algorithm 5. EC^2Sky structure construction

input : E: Dataset, D: Data space of E, S: Subspace with static preferences, Z: Subspace with dynamic preferences
output: EC^2Sky structure

```
// Step 1: computation of static and order sensitive skyline
   points
```
1 Compute $GSky(D, E)$
2 Store $GSky(D, E)$ in the EC^2Sky structure
3 Compute $OsSky(D, E)$
4 Store $OsSky(D, E)$ in the EC^2Sky structure

```
// Step 2: computation of the EC2Sky structure
```
5 **foreach** $d_i \in Z$ **do**
6 Compute and Store CP_{d_i} in the EC^2Sky structure
7 **foreach** $\wp_i^j \in \wp_i$ **do**
8 Compute $NewSky_{(d_i,\wp_i^j)}$; `// Algorithm 1`
9 Store $NewSky_{(d_i,\wp_i^j)}$ in the EC^2Sky structure
10 Compute $CandComp_{(d_i,\wp_i^j)}$; `// Algorithm 2`
11 Store $CandComp_{(d_i,\wp_i^j)}$ in the EC^2Sky structure

when a new dimension is introduced, while new skyline points (*CompSky* points) may appear.

The computation of the *CutSky* set is described by Algorithm 3. We just show how to compute $CutSky(D^{i-1})$ because the calculation of $CutSky(d_i \bigcup S)$ is performed in the same way.

The compromise skyline points are the set of compromise candidates that become skyline compromises. The computation of this set is described by Algorithm 4.

We are now in position to detail the EC^2Sky method. Algorithm 5 and 6 describe the general process of EC^2Sky. Algorithm 5 outlines the required steps to construct the EC^2Sky structure. At the end of step 2 (Algorithm 5, lines 8 and 11), we calculate the skyline sets, stored in the EC^2Sky structure. Each of these sets corresponds to one dimension with dynamic preferences defined in the user query.

Algorithm 6 is dedicated to step 3, the computation of changing elements of the skyline. The sets *CompSky* (Algorithm 6, line 11) and the union of *CutSky* (Algorithm 6, line 10) are computed. As stated by the *incremental skyline theorem* (Theorem 2), the final skyline is obtained by eliminating all the *CutSky* points and by adding all the *CompSky* points to the union of skylines related to queries involving one dynamic preference (Algorithm 6, line 12).

Algorithm 6. $EC^2Sky(Sky(D,E)_{(Z,\wp)})$

input : $Sky(D,E)_{(Z,\wp)}$:skyline query
output: $Sky(D,E)_{(Z,\wp)}$

// Step 3: computation of changing points
$Sky(D^0,E)_{(Z,\wp)} \leftarrow GSky(D,E)$
for $i \leftarrow 1$ **to** $m = |Z|$ **do**
  if $\wp_i = \emptyset$ **then**
    $Sky(d_i \bigcup S,E)_{(Z,\wp)} \leftarrow GSky(D,E) \bigcup CP_{d_i}$
  else
    if *isFirstOrderPref*($\wp_i$) **then**
      $Sky(d_i \bigcup S,E)_{(Z,\wp)} \leftarrow GSky(D,E) \bigcup NewSky_{(d_i,\wp_i^j)}$
    else
      Use the merging property
  Compute $CutSky(D^{i-1})$ (resp. $CutSky(d_i \bigcup S)$)
  Compute $NewCompSky(D^i,E)$; // Algorithm 4
  $Sky(D^i,E)_{(Z,\wp)} \leftarrow Sky(D^{i-1},E)_{(Z,\wp)} \bigcup Sky(d_i \bigcup S,E)_{(Z,\wp)} \bigcup NewCompSky(D^i,E) - (CutSky(D^{i-1}) \bigcup CutSky(d_i \bigcup S))$
$Sky(D,E)_{(Z,\wp)} \leftarrow Sky(D^m,E)_{(Z,\wp)}$

Example 21. *The skyline associated with the preferences* $\wp = \{M <_{Gr} H <_{Gr} *, G <_{Air} *\}$*, can be computed from the skyline associated with the preferences* $\wp_1 = \{M <_{Gr} H <_{Gr} *\}$ *and* $\wp_2 = \{G <_{Air} *\}$.
Let $D^1 = \{Price, Distance, Gr\}$ *and* $D^2 = \{Price, Distance, Gr, Air\}$*. The skyline associated to* $\wp_1$ *and* $\wp_2$ *is computed in the same way as in example 20.*
$Sky(D^1,E)_{(Z,M<_{Gr}H<_{Gr}*)} = \{a,b,e,f,h\}$ *and* $Sky(\{Air\} \bigcup S,E)_{(Z,G<_{Air}*)} = \{a,b,e\}$.

Since, we have two dimensions with dynamic preferences we compute the sets $CutSky(D^1)$*,* $CutSky(\{Air\} \bigcup S)$ *and* $NewCompSky(D^2,E)$*. For this example,* $CutSky(D^1) = \emptyset$*,*
$CutSky(\{Air\} \bigcup S) = \emptyset$ *and* $NewCompSky(D^2,E) = \emptyset$*. Finally,*
$Sky(D^2,E)_{(Z,\wp)} = Sky(D^1,E)_{(Z,M<_{Gr}H<_{Gr}*)} \bigcup Sky(\{Air\} \bigcup S,E)_{(Z,G<_{Air}*)}$
$= \{a,b,e,f,h\}$.

Our proposal provides the user with a way to express preferences and with the ability to change them without being penalized by long response times. A good performance is achieved by storing only the minimal amount of information required to enable quick and easy updates.

The experimental evaluation presented in the following highlight the relevance of the proposed solution.

6 Experiments

In this section, we report an experimental evaluation of our algorithm EC^2Sky on synthetic data sets. EC^2Sky is implemented in $C/C++$ and the experiments were performed on an Intel Xeon CPU at 3GHz and 16 GB of RAM on a Linux platform. For static dimensions, the data were produced by the generator released by the authors of [4]. Three kinds of data sets were generated: independent data, correlated data and uncorrelated data. The description of these data sets can be found in [4]. Like in [20], we only show the experimental results for the uncorrelated data sets. The results for independent data sets and correlated data sets are similar, but the execution times are much shorter for correlated data sets. The dynamic dimensions were generated according to a Zipfian distribution [19]. By default, we set the Zipfian θ parameter to 1. We obtained $1,000,000$ tuples for 6 dimensions with static order. The number of dynamic dimensions varied from 1 to 40 and the cardinality of these dimensions varied from 2 to 50. We chose a query template such that the most frequent value of some dynamic dimension has the highest priority over all other values. This represents a parameter that becomes more difficult to manage as the skyline tends to be larger.

Table 4. Default values

Parameter	Value
No. of tuples	100K
No. of dimensions with static preferences (prefs)	6
No. of dimensions with dynamic prefs	4
No. of values in dimension with dynamic prefs	4
Zipfian parameter θ	1

In the following experiments, we compare the performance of our algorithm EC^{2}Sky with the algorithm IPO-tree implemented by [20], in terms of the execution time and the storage size.

Scalability with Respect to Dimensionality. In the first experiments, the number of static dimensions was set to 6 and the number of dynamic dimensions varied from 1 to 40. Figure 1 shows that the execution time and the storage size of both EC^2Sky and IPO-tree increase with the number of dynamic dimensions. However, the increase rate of IPO-tree is greater than the increase rate of EC^2Sky. This comes from the complexity of the preferences tree built by IPO-tree. The IPO-tree structure contains more nodes, yielding a larger storage size. Beyond 6 dynamic dimensions, IPO-tree overflows the available memory. This is due to its tree size in $O(c^m)$ (m is the number of dynamic dimensions and c the cardinality of a dimension), which induces an exponential increase of the storage size. The table built by EC^2Sky has a size in $O(c * m)$, which induces a

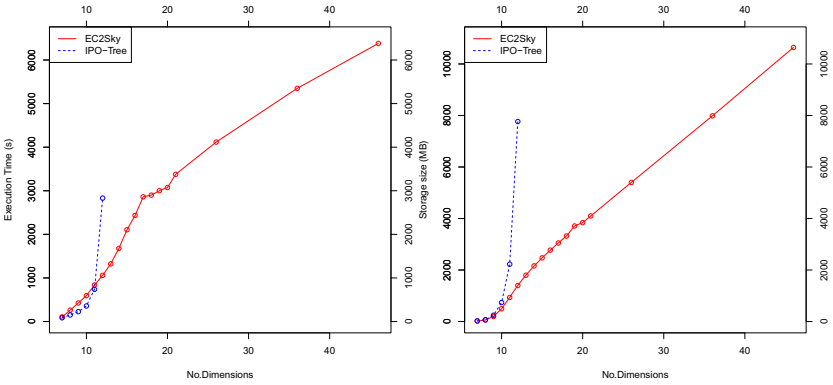


Fig. 1. Scalability with respect to dimensionality

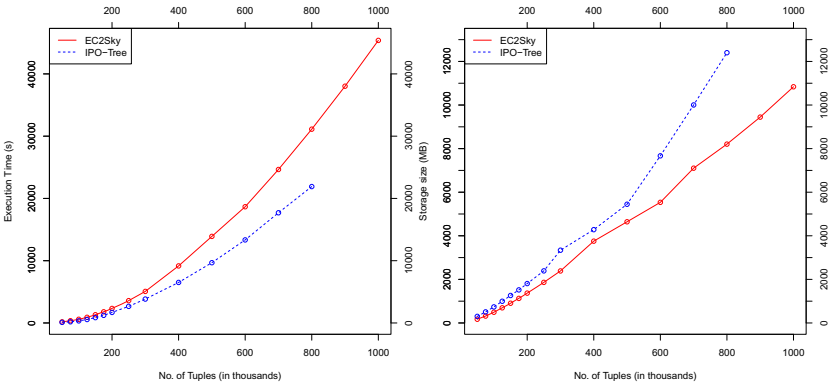
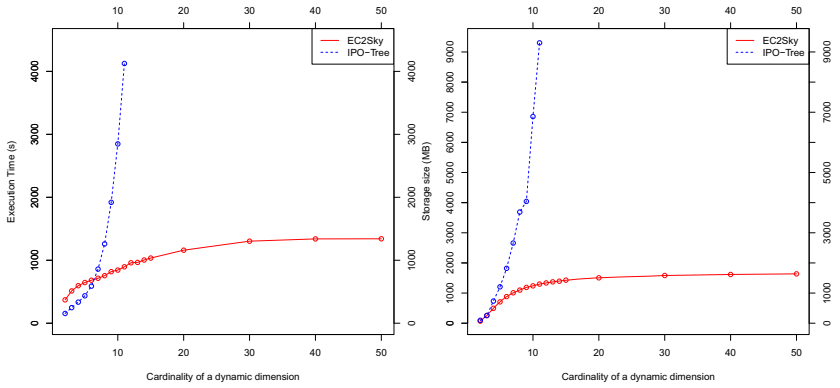


Fig. 2. Scalability with respect to database size

substantial increase of the storage size but which evolves more slowly than the IPO-tree. The results are similar to those in Figure 1 when the number of static dimensions is set to 4.

Scalability with Respect to the Database Size. In this experiment, the number of tuples of the dataset varies from $50,000$ to $1,000,000$. Figure 2 shows that the execution time and the storage size of both EC^2Sky and IPO-tree increase with the size of the dataset. This is because the size of the information stored and analyzed increases with the increase of the dataset. However, our method is more efficient than the IPO-tree method. Beyond 800.000 tuples, IPO-tree overflows the available memory. Indeed, IPO-tree stores all the skylines



(a) (b)

Fig. 3. Scalability with respect to the cardinality of the dynamic dimensions

associated to all possible combinations of the different first order preferences of all the dynamic dimensions whereas EC^2Sky stores only the skyline points corresponding to the first order preferences of each dynamic dimension. The skyline of the various combinations of preferences are derived from simple operations of intersection and union.

Scalability with Respect to the Cardinality of Dynamic Dimensions. We vary the cardinality of the dynamic dimensions from 2 to 50. Figure 3 shows that the execution time and the storage size of both EC^2Sky and IPO-tree increase when the cardinality of the dimensions increases. Once more, EC^2Sky is more efficient than IPO-tree. The size of the treestructure of IPO-tree is exponential in $O(c^m)$. So, it becomes more complex and larger when the cardinality of dimensions (c) increases. IPO-tree overflows the available memory for a dynamic dimension cardinality above 11. We can also observe a significant increase of the related execution time.

7 Conclusion

In this paper, we have proposed a new efficient method to compute skyline queries in the presence of dimensions associated with dynamic user preferences. We have investigated preferences on dimension values that can be expressed by any partial or complete order, with a particular focus on the compromise points which are important in decision making. Our approach, is based on a materialization of the first order preferences, that can respond efficiently to skyline queries related to user preferences even in the context of large volumes of data. The experimentations presented in this paper highlight the performance improvements of EC^2Sky compared to IPO-tree [20].

The consideration of dimensions with dynamic preferences opens several promising future research directions. First, to demonstrate the usefulness of our method, we want to experiment our algorithm on a real data set. We are particularly interested in the analysis of simulation results from a biophysical model to extract the most polluting plots in a watershed with respect to different analysis criteria. Another possible future direction is to investigate how to compute skyline queries in the context of hierarchical and aggregated data. The adopted approach would search the best compromises along the set of axes. However, this approach raises several problems. One is to define a computation adapted to the level of the explored hierarchy. Another is to define the semantics of skyline points at different levels of granularity.

Acknowledgments. This work was funded by the French National Research Agency (ANR) through the ACASSYA project (ANR-08-STRA-01).

References

[1] Balke, W.T., Guntzer, U., Siberski, W.: Exploiting indifference for customization of partial order skylines. In: Proceedings of the 10th International Database Engineering and Applications Symposium, pp. 80–88. IEEE Computer Society (2006)
[2] Bentley, J.L., Kung, H.T., Schkolnick, M., Thompson, C.D.: On the average number of maxima in a set of vectors and applications. J. ACM 25(4), 536–543 (1978)
[3] Bitran, G.R., Magnanti, T.L.: The structure of admissible points with respect to cone dominance. Optimization Theory and Applications 29(4), 573–614 (1979)
[4] Borzsonyi, S., Kossmann, D., Stocker, K.: The skyline operator. In: Proc. of the 17th International Conference on Data Engineering, pp. 421–430. IEEE Computer Society (2001)
[5] Bouadi, T., Cordier, M.-O., Quiniou, R.: Incremental computation of skyline queries with dynamic preferences. In: Liddle, S.W., Schewe, K.-D., Tjoa, A.M., Zhou, X. (eds.) DEXA 2012, Part I. LNCS, vol. 7446, pp. 219–233. Springer, Heidelberg (2012)
[6] Brando, C., Goncalves, M., González, V.: Evaluating top-k skyline queries over relational databases. In: Wagner, R., Revell, N., Pernul, G. (eds.) DEXA 2007. LNCS, vol. 4653, pp. 254–263. Springer, Heidelberg (2007)
[7] Chen, L., Lian, X.: Efficient processing of metric skyline queries. IEEE Trans. on Knowl. and Data Eng. 21(3), 351–365 (2009)
[8] Chomicki, J., Godfrey, P., Gryz, J., Liang, D.: Skyline with presorting: Theory and optimizations. In: Proc. of Intelligent Information Systems, pp. 595–604. Springer, Heidelberg (2005)
[9] Godfrey, P., Shipley, R., Gryz, J.: Algorithms and analyses for maximal vector computation. The VLDB Journal 16(1), 5–28 (2007)
[10] Huang, Z., Guo, J., Sun, S.L., Wang, W.: Efficient optimization of multiple subspace skyline queries. J. Comput. Sci. Technol. 23(1), 103–111 (2008)
[11] Jin, W., Tung, A.K.H., Ester, M., Han, J.: On efficient processing of subspace skyline queries on high dimensional data. In: Proc. of the 19th International Conference on Scientific and Statistical Database Management. IEEE Computer Society (2007)

[12] Mindolin, D., Chomicki, J.: Preference elicitation in prioritized skyline queries. The VLDB Journal 20(2), 157–182 (2011)
[13] Papadias, D., Tao, Y., Fu, G., Seeger, B.: Progressive skyline computation in database systems. ACM Trans. Database Syst. 30(1), 41–82 (2005)
[14] Pei, J., Jin, W., Ester, M., Tao, Y.: Catching the best views of skyline: a semantic approach based on decisive subspaces. In: Proc of the 31st International Conference on Very Large Data Bases, pp. 253–264, VLDB Endowment (2005)
[15] Raïssi, C., Pei, J., Kister, T.: Computing closed skycubes. Proc. VLDB Endow. 3(1), 838–847 (2010)
[16] Sawaragi, Y., Nakayama, H., Tanino, T.: Theory of Multiobjective Optimization. Academic Press, Orlando (1985)
[17] Tan, K.L., Eng, P.K., Ooi, B.C.: Efficient progressive skyline computation. In: Proceedings of the 27th International Conference on Very Large Data Bases, pp. 301–310. Morgan Kaufmann Publishers Inc. (2001)
[18] Tao, Y., Xiao, X., Pei, J.: Efficient skyline and top-k retrieval in subspaces. IEEE Trans. on Knowl. and Data Eng. 19(8), 1072–1088 (2008)
[19] Trenkler, G.: In: Johnson, N.l., Kotz, S., kemp, A.W. (eds.) Univariate Discrete Distributions, 2nd edn. John wiley (1994) ISBN 0-471-54897-9; Computational Statistics & Data Analysis, 17(2), 240–241 (1994)
[20] Wong, R.C.W., Fu, A.W.C., Pei, J., Ho, Y.S., Wong, T., Liu, Y.: Efficient skyline querying with variable user preferences on nominal attributes. Proc. VLDB Endow. 1(1), 1032–1043 (2008)
[21] Wong, R.C.W., Pei, J., Fu, A.W.C., Wang, K.: An erratum on "online skyline analysis with dynamic preferences on nominal attributes". IEEE Trans. on Knowl. and Data Eng. (to be published)
[22] Wong, R.C.W., Pei, J., Fu, A.W.C., Wang, K.: Online skyline analysis with dynamic preferences on nominal attributes. IEEE Trans. on Knowl. and Data Eng. 21(1), 35–49 (2009)
[23] Xia, T., Zhang, D., Tao, Y.: On skylining with flexible dominance relation. In: Proc. of the 2008 IEEE 24th International Conference on Data Engineering, pp. 1397–1399. IEEE Computer Society (2008)
[24] Yuan, Y., Lin, X., Liu, Q., Wang, W., Yu, J.X., Zhang, Q.: Efficient computation of the skyline cube. In: Proc. of the 31st International Conference on Very Large Data Bases, pp. 241–252, VLDB Endowment (2005)

The Finite Implication Problem for Expressive XML Keys: Foundations, Applications, and Performance Evaluation

Flavio Ferrarotti[1], Sven Hartmann[2], Sebastian Link[3], Mauricio Marin[4], and Emir Muñoz[5]

[1] Victoria University of Wellington
flavio.ferrarotti@vuw.ac.nz
[2] Clausthal University of Technology
sven.hartmann@tu-clausthal.de
[3] The University of Auckland
s.link@auckland.ac.nz
[4] Yahoo! Research
mmarin@yahoo-inc.com
[5] DERI, National University of Ireland Galway
emir.munoz@deri.org

Abstract. The increasing popularity of XML for persistent data storage, processing and exchange has triggered the demand for efficient algorithms to manage XML data. Both industry and academia have long since recognized the importance of keys in XML data management. In this paper we make a theoretical as well as a practical contribution to this area. This endeavour is ambitious given the multitude of intractability results that have been established. Our theoretical contribution is based in the definition of a new fragment of XML keys that keeps the right balance between expressiveness and efficiency of maintenance. More precisely, we characterize the associated implication problem axiomatically and develop a low-degree polynomial time decision algorithm. In comparison to previous work, this new fragment of XML keys provides designers with an enhanced ability to capture properties of XML data that are significant for the application at hand. Our practical contribution includes an efficient implementation of this decision algorithm and a thorough evaluation of its performance, demonstrating that reasoning about expressive notions of XML keys can be done efficiently in practice, and scales well. Our results promote the use of XML keys on real-world XML practice, where a little more semantics makes applications a lot more effective. To exemplify this potential, we use the decision algorithm to calculate non-redundant covers for sets of XML keys. In turn, this allow us to reduce significantly the time required to validate large XML documents against keys from the proposed fragment.

1 Introduction

Keys are the most important class of integrity constraints used in database management. First of all, they provide a mechanism for identifying objects in

A. Hameurlain et al. (Eds.): TLDKS X, LNCS 8220, pp. 60–94, 2013.

database instances, thus establishing an invariant relationship between objects in the real world and their representation in the database. The increasing popularity of XML [6] for persistent data storage and data processing has triggered the demand for efficient algorithms to manage XML data. Both industry and academia have long since recognized the importance of keys in XML data management. Over the last decade, several notions of XML keys have been proposed and discussed in the database community. (See [14] for a brief overview). The most influential proposal is due to Buneman et al. [7,8] who defined keys on the basis of an XML tree model similar to the one suggested by DOM [3] and XPath [10]. Figure 1 shows such a representation in which nodes are annotated by their type: E for element nodes, A for attribute nodes, and S for text nodes (PCDATA). While Buneman et al. studied keys as a concept orthogonal to schema specification (such as DTD or XSD), their proposal has been adopted by the W3C for the XML Schema standard [23] subject to some minor, though essential modifications (see [4] for a discussion). Today, all major XML-enabled DBMSs, XML parsers and editors (such as XMLSpy) support keys.

The XML keys considered in this work uniquely identify nodes in an XML tree by (complex) values on some selected descendant nodes. These keys are defined using path expressions to select the relevant sets of nodes. In Figure 1, a clear example of a key is that a *bank* node can be uniquely identified by the value of its *name* attribute. This is an instance of an absolute key which is satisfied if it holds globally in the entire XML tree. We also work with relative keys, which are satisfied if they hold locally within some subtrees. For instance, in every subtree rooted at an *account* node, a *transaction* node can be uniquely identified by the value of its child node, which can either be a *withdrawal* node or a *deposit* node. This XML key might not hold in the entire tree since there might well be different withdrawals or deposits made at the same time, on the same day and for the same amount, as long as they do not belong to the same account. We emphasize that the element nodes *withdrawal* and *deposit* have complex content. Thus, checking whether two *transaction* nodes violate this latter key involves testing whether the subtrees rooted at their respective child nodes are isomorphic to one another, with the identity on string values.

1.1 Motivation

For relational data, keys have been widely used to improve the performance of many perennial tasks in database management, ranging from consistency checking to query answering. The hope is that keys will turn out to be equally beneficial for XML. One of the most fundamental questions on keys is that of logical implication, that is, deciding if a new key holds given a set of known keys. If the implication of XML keys can be decided efficiently, then it is possible to take advantage not only of those keys that were specified explicitly by the database designer, but also of those ones that were specified implicitly. Among other things, this is important for minimizing the cost of validating that an

⟨**bank name=** *"ANZ"*⟩⟨**branch name=** *"Downtown"*⟩
⟨**client cno=** *"JOH23144"*⟩⟨**account no=** *"2144964"*, **kind=** *"savings"*⟩
⟨**transaction**⟩⟨**withdrawal**⟩
⟨**date**⟩ *"2012.02.04"*⟨**/date**⟩⟨**time**⟩ *"13:01:03"*⟨**/time**⟩⟨**amount**⟩ *"$500.00"*⟨**/amount**⟩
⟨**/withdrawal**⟩⟨**/transaction**⟩
⟨**transaction**⟩⟨**deposit**⟩
⟨**date**⟩ *"2012.02.04"*⟨**/date**⟩⟨**time**⟩ *"14:15:03"*⟨**/time**⟩⟨**amount**⟩ *"$302.34"*⟨**/amount**⟩
⟨**/deposit**⟩⟨**/transaction**⟩
⟨**/account**⟩⟨**/client**⟩
⟨**/branch**⟩⟨**/bank**⟩
⟨**bank name=** *"BNZ"*⟩⟨**branch name=** *"Northshore"*⟩
...
⟨**/branch**⟩⟨**/bank**⟩

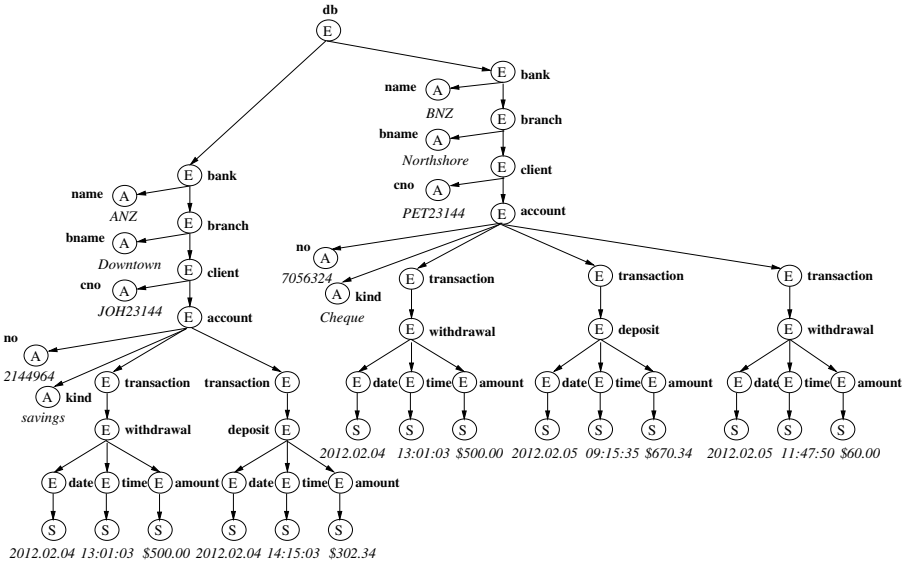


Fig. 1. XML data fragment and its tree representation

XML document satisfies a set of keys gathered as business rules during requirements engineering.

Example 1. Suppose, a database designer has already specified an XML key φ_a which expresses that in every subtree rooted at a *bank* node, a *client* node can be uniquely identified by its child attribute *cno* together with its child node *account*. Later on, another of the database designers discovers that the *cno* attribute can by itself be used to identify a *client* node relatively to a *bank* node, and thus defines a new suitable XML key φ_b. It is easy to see that if φ_b is satisfied by an XML tree T, then also the former key φ_a is satisfied by T. That is, φ_b implies φ_a. Thus, we can validate an XML tree T against both keys by just checking whether T satisfies φ_b, since T will satisfy both φ_a and φ_b iff it satisfies φ_b. We would like to emphasize that the *account* nodes have complex content. Thus, checking whether two *client* nodes in T violate φ_a is quite costly in terms

of time, since it involves testing whether the subtrees rooted at their *account* nodes are isomorphic to one another, with the identity on string values. In contrast, checking whether two *client* nodes violate φ_b only involves checking equality on the text nodes of their respective *cno* attributes. □

No less important, facilities for reasoning about XML keys present numerous opportunities for designing XML databases and views that permit a more efficient processing of frequent queries and updates.

Example 2. Let us consider the following XQuery, which expresses over XML trees with the structure of the tree in Figure 1, a client request for her transactions on an specific account and date.

```
for $a in doc("transactions.xml")//account[@no]="2144964"
where $a//date/text() ="2012.02.04"
return ⟨transactions⟩{$a/transaction/*}⟨/transactions⟩
```

Also, let us assume that this is the most frequent type of query. It is expensive to process an XQuery such as this one over an XML tree with the layout of the tree in Figure 1, since for all *transaction* nodes of the selected account, it has to be decided if it has a descendant *date* node which corresponds to the specified date. This is due to the fact that there can be many transactions on the selected account for the given date, and thus a *transaction* node cannot be uniquely identified by its descendant *date* node. Note that this key is therefore not implied by the keys that apply to the XML tree. Also note that if the *transaction* nodes are encrypted, then the evaluation of the query requires us to decrypt every single one of them for the selected account. In this case, the layout illustrated in Figure 2 represents a better choice. The target *transaction* nodes are now grouped together as child nodes of a target *txdate* node. Redundancies with respect to *dates* for a given account are eliminated, which results in a better overall design. Moreover, only those *transaction* nodes which are child nodes of the selected *txdate* node need to be decrypted. Note that these improvements are due to the existence of a new relative key which specifies that, in every subtree rooted at an *account* node, a *txdate* node can be uniquely identified by the value of its *date* attribute. The original XQuery is rewritten into

```
for $a in doc("transactions.xml")//account[@no]="2144964"
where $a/txdate[@date] ="2012.02.04"
return ⟨transactions⟩{$a/txdate/transaction/*}⟨/transactions⟩
```

□

A further example of an area in which the implication of XML keys is of tremendous benefit is semantically rich data exchange.

Example 3. Suppose we need to share part of the information on bank transactions with the Reserve Bank, and that due to legal requirements we need to eliminate the information regarding clients and individual accounts. Thus, we

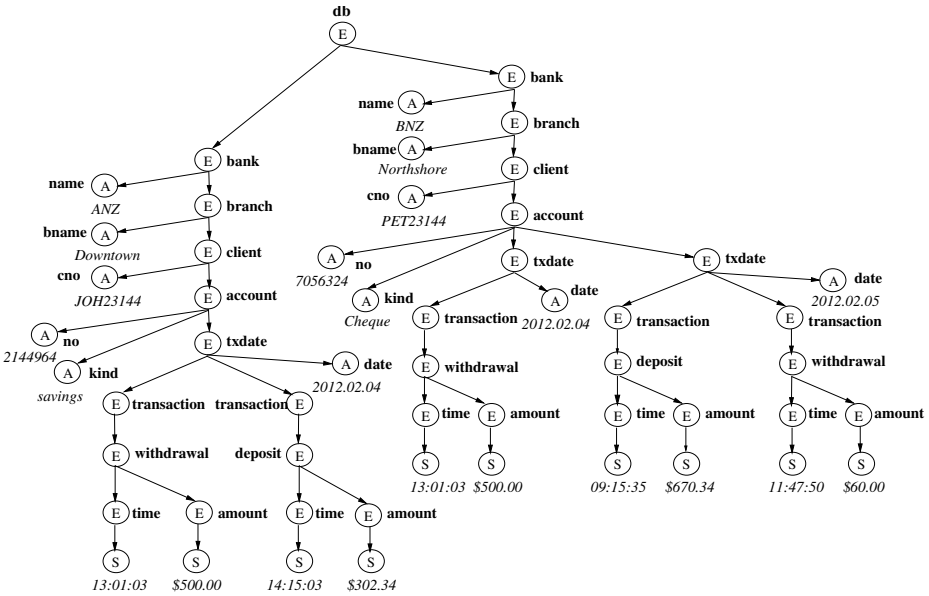


Fig. 2. XML data fragment with improved layout

generate a view over the XML tree which skips the *no* attribute of the *account* nodes and the *cno* attribute of the *client* nodes. Clearly, it would be useful to provide the Reserve Bank with the set of keys defined for the original XML tree. In light of this additional semantic information, they could better interpret the XML tree. For instance, they could deduce from those keys that a *client* node was identifiable in the original XML document by a *cno* attribute relatively to a *bank* node. Furthermore, they could check whether the specified keys allow one to conclude further keys which are useful for extraction and processing of the data in the provided XML document. For instance, they could check whether the specified keys allow one to conclude a further key stating that a *client* node can be identified by its descendant *account* nodes relatively to a *bank* node. □

1.2 Related Work

The definition of keys, adopted by the W3C for XML Schema [23], is currently the industry standard for specifying keys. However, Arenas et al. [4] have shown the computational intractability of the associated consistency problem, i.e., the question whether there exists an XML document that conforms to a given XSD and satisfies the specified keys. A further issue pointed out by Buneman et al. [7] is the fact that XML Schema restricts value equality to string-valued data items. But there are cases in which keys are not so restricted (see Section 7.1 of [7] for discussion). In particular, we have given examples of XML keys that require

a less restricted notion of equality, since they require us to test equality on element nodes such as *withdrawal* and *deposit* which are not string-valued. On the other hand, the expressiveness and computational properties of XML keys with good reasoning capabilities have been deeply studied from a theoretical perspective [7,8,15,16,11].

In practice, however, expressive yet tractable notions of XML keys have been mostly ignored. Even though several algorithms that validate XML documents against sets of certain XML keys have been proposed and tested with promising results (see e.g. [9,18]), none of them make use of the reasoning capabilities of XML keys as suggested in Example 1.

Aiming to fill this gap between theory and practice, we initiated in [12] an empirical study of an XML key fragment, namely the fragment of XML keys with nonempty sets of simple key paths. As shown in [8,15], automated reasoning about this XML key fragment can be done efficiently, in theoretical terms. Our work confirmed this fact in practice. This article extends these earlier works by considering a strictly more expressive fragment of XML keys which allows the use of single- and variable-length wildcards in the path language used to specify the keys, as well as empty sets of key paths for the specification of structural keys which identify nodes (within subtrees) by unique paths instead of unique data value. In fact, we establish a fragment that strictly includes the already expressive fragments of XML keys explored in [16,11] without resigning efficiency.

1.3 Contributions

Evidently, the expressiveness (but also the tractability) of a fragment of XML keys will depend on the query language used to navigate between nodes in trees. As common in XML data processing, path expressions are used to select the nodes of interest. The preferred languages for selection queries against XML data are XPath queries [10] and fragments thereof. The flexibility provided by these languages is probably not needed for every application, but there are many applications where flexibility is essential. Taken that XML is widely used to represent heterogeneous data, the expressiveness that comes with wildcards and descendant queries is highly appreciated. One may think of data integration where data from different sources are stored in the same document. Though not uniformly structured, the data should still be analyzed using the same 'one fits all' query to avoid extensive maintenance costs. This can only be achieved when using a sufficiently rich query language.

Thus, our first contribution establishes a new fragment of XML keys (namely Max-Keys) that keeps the right balance between expressiveness and efficiency of maintenance. More precisely, we characterize the associated implication problem axiomatically, and propose a low-degree polynomial time decision algorithm. The set of XML keys that are expressible in this new fragment includes strictly the sets of XML keys that are expressible in the fragments considered in

previous proposals [15,16,11,12]. The first source of expressiveness results from the very general notion of value equality: two element nodes v and w are considered value equal, if the subtrees rooted at v and w are isomorphic by an isomorphism that is the identity on string values. This contrasts with more restrictive notions, for instance with value equality on leaf nodes. The second source of expressiveness is a result of the path language we use to select nodes. This includes the single-label wildcard, child navigation, and descendant navigation as known from XPath. Note that in [12] we have considered XML keys whose context and target nodes can be selected by a path language that uses child navigation and descendant navigation, and key nodes by using only child navigation. While this fragment of XML keys can already capture many desirable properties, the Max-Keys fragment proposed in this paper captures a considerably more expressive class of properties by allowing the use of single-label wildcards, a controlled use of variable-length wildcards in the key paths and XML keys without key paths that allow one to express structural keys.

Our second contribution concerns the exploration of this theoretical ideas in practice. We develop and implement an efficient algorithm that decides the implication problem for Max-Keys and thoroughly evaluate its performance. Our performance tests give empirical evidence that reasoning about expressive notions of XML keys is practically efficient, and scales well. In fact, the performance of this new algorithm is comparable to the performance of the algorithm presented in [12] for a strictly less expressive fragment of XML keys. Our results unleash XML keys on real-world XML practice, where a little more semantics makes applications a lot more effective.

There is indeed great potential for practical uses of the proposed decision algorithm. For instance, the process of checking XML data integrity against XML keys can benefit a lot from the ability to decide implication efficiently. Clearly, if a set Σ of XML keys implies an XML key φ, and we have already checked that an XML data tree satisfies Σ, then there is no need to test φ any more, saving considerable resources. Thus, exploiting our algorithm we compute non-redundant covers for sets of XML keys. A set Σ of keys is non-redundant if there is no key σ in Σ such that σ is implied by $\Sigma - \{\sigma\}$. Same than for the less expressive fragment of XML keys studied in [12], our experiments show that the time to compute a non-redundant cover for a given set of keys in the fragment of Max-Keys, is just a small fraction of the average time needed to validate an XML document against a single key.

We would like to note that the experiments carried out in this article extend the experiments presented in [12] in several ways. Firstly, we consider XML keys which are more expressive than the keys studied in our previous paper. Secondly, we test the new implication algorithm by considering even larger sets of XML keys (of up to 180 keys). Thirdly, we include a new XML document which holds the complete Chilean electoral roll in our validation experiments. This XML document, which contains 3.2GB of data, is several orders of magnitude larger than the XML documents usually considered in previous works in this area.

Finally, we compute all non-redundant covers that are subsets of a given set of XML keys, and rank them according to the time it takes for their validation. This provides valuable information which can be used in practice to design XML documents which are more efficient to validate. For replication purposes, all the data sets used in our experiments as well as the full set of results can be downloaded from `http://emir-munoz.github.io/xml-constraints`.

1.4 Organization

We recall basic concepts and fix the formal notation in Section 2. In Section 3 we define the central notion of XML key and introduce a new and expressive fragment of XML keys (namely the Max-Keys). We establish a finite set of inference rules for deriving new Max-Keys in Section 4 and prove its completeness in Section 5. In Section 6, we presents an efficient algorithm for deciding implication of Max-Keys as well as an implementation thereof. We present the results regarding the application and performance evaluation of our decision algorithm for the finite implication problem of Max-Keys, as well as an strategy to use this algorithm in the context of XML document validation, in Section 7. We conclude the paper in Section 8 with final remarks.

2 Preliminaries

2.1 XML Data Representation

We use the common representation of XML data as ordered, node-labeled trees.

Let $\mathbf{E}$ denote a countably infinite set of element tags, $\mathbf{A}$ a countably infinite set of attribute names, and $\{S\}$ a singleton set denoting text (PCDATA). These sets are pairwise disjoint. The elements of $\mathcal{L} = \mathbf{E} \cup \mathbf{A} \cup \{S\}$ are called *labels*.

An *XML tree* is a 6-tuple $T = (V, lab, ele, att, val, r)$ where V is a set of nodes, and lab is a mapping $V \to \mathcal{L}$ assigning a label to every node in V. A node $v \in V$ is an *element node* if $lab(v) \in \mathbf{E}$, an *attribute node* if $lab(v) \in \mathbf{A}$, and a *text node* if $lab(v) = S$. Moreover, ele and att are partial mappings defining the edge relation of T: for any node $v \in V$, if v is an element node, then $ele(v)$ is a list of element and text nodes, and $att(v)$ is a set of attribute nodes in V. If v is an attribute or text node, then $ele(v)$ and $att(v)$ are undefined. The partial mapping val assigns a string to each attribute and text node: for each node $v \in V$, $val(v)$ is a string if v is an attribute or text node, while $val(v)$ is undefined otherwise. Finally, r is the unique and distinguished root node.

For a node $v \in V$, each node w in $ele(v)$ or $att(v)$ is called a *child* of v, and we say that there is an edge (v, w) from v to w in T. A *path* p of T is a finite sequence of nodes $v_0, \ldots, v_m$ in V such that (v_{i-1}, v_i) is an edge of T for $i = 1, \ldots, m$. The path p determines a word $lab(v_1).\cdots.lab(v_m)$ over the alphabet $\mathcal{L}$, denoted by $lab(p)$. For a node $v \in V$, each node w reachable from v is called a *descendant* of v. Note that every XML tree has a tree structure: for each node $v \in V$, there is a unique path from the root node r to v.

2.2 Value Equality of Nodes on XML Trees

Two nodes $u, v \in V$ are *value equal*, denoted by $u =_v v$, iff the subtrees rooted at u and v are isomorphic by an isomorphism that is the identity on string values. More formally, $u =_v v$ whenever the following conditions are satisfied:

a. $lab(u) = lab(v)$.
b. If u, v are attribute or text nodes, then $val(u) = val(v)$.
c. If u, v are element nodes, then *(i)* if $att(u) = \{a_1, \ldots, a_m\}$, then $att(v) = \{a'_1, \ldots, a'_m\}$ and there is a permutation π on $\{1, \ldots, m\}$ such that $a_i =_v a'_{\pi(i)}$ for $i = 1, \ldots, m$, and *(ii)* if $ele(u) = [u_1, \ldots, u_k]$, then $ele(v) = [v_1, \ldots, v_k]$ and $u_i =_v v_i$ for $i = 1, \ldots, k$.

Note that the notion of value equality takes the document order of the XML tree into account. We remark that $=_v$ is an equivalence relation on the node set V of the XML tree. As an example, the first and third *transaction* nodes (from left to right) in Figure 1 are value equal while all the remaining *transaction* nodes are not value equal to each other.

2.3 Node Selection Queries on XML Trees

Regular paths have been widely used to express queries for selecting nodes in XML trees. In the sequel, we use the path language $PL^{\{\cdot,\_,\_^*\}}$ consisting of expressions given by the following grammar:

$$Q \to \ell \mid \varepsilon \mid Q.Q \mid \_ \mid \_^*$$

Herein, $\ell \in \mathcal{L}$ is any label, ε denotes the empty path expression, "." denotes the concatenation of two path expressions, "_" denotes the *single-label* wildcard, and "_*" denotes the *variable-length* wildcard.

Let P, Q be words from $PL^{\{\cdot,\_,\_^*\}}$. P is a *refinement* of Q, denoted by $P \lesssim Q$, if P is obtained from Q by replacing variable-length wildcards in Q by words from $PL^{\{\cdot,\_,\_^*\}}$ and single-label wildcards in Q by labels from $\mathcal{L}$. For example, *bank.branch.account* is a refinement of *bank._.account*. Note that $\lesssim$ is a pre-order on $PL^{\{\cdot,\_,\_^*\}}$. Let $\sim$ denote the congruence induced by the identity $\_^*.\_^* = \_^*$ on $PL^{\{\cdot,\_,\_^*\}}$, and observe that $P \sim Q$ holds if and only if P and Q are refinements of each other. For example, *bank._*.account* $\sim$ *bank._*._*.account*.

Regular paths allow one to navigate in an XML tree. We briefly recall the semantics of expressions from $PL^{\{\cdot,\_,\_^*\}}$ in the context of XML. Let Q be a word from $PL^{\{\cdot,\_,\_^*\}}$. A path p in the XML tree T is called a Q-path if $lab(p)$ is a refinement of Q. For nodes $v, w \in V$, we write $T \models Q(v, w)$ if w is reachable from v following a Q-path in T.

For a node v in the XML tree T, let $v[\![Q]\!]$ denote the set of nodes in T that are reachable from v following any Q-path, that is, $v[\![Q]\!] = \{w \mid T \models Q(v, w)\}$. In particular, we use $[\![Q]\!]$ as an abbreviation for $r[\![Q]\!]$ where r is the root node.

For a subset $\mathcal{Z} \subseteq \{., \_, \_^*\}$, let $PL^{\mathcal{Z}}$ denote the subset of $PL^{\{\cdot,\_,\_^*\}}$ with expressions restricted to the constructs in $\mathcal{Z}$. In particular, $PL^{\{\cdot\}}$ is the set of simple path expression without wildcards.

Since attribute and text nodes in an XML tree T are always leaves, $Q \in PL^{\{\cdot,\_,\_^*\}}$ is *valid* only if it has no labels $\ell \in \mathbf{A}$ or $\ell = S$ in a position other than the terminal one. Note that each prefix of a valid Q is valid, too.

Let P, Q be words from $PL^{\{\cdot,\_,\_^*\}}$. P is *contained* in Q, denoted by $P \subseteq Q$, if for every XML tree T and every node v of T we have $v[\![P]\!] \subseteq v[\![Q]\!]$. It follows immediately from the definition that $P \lesssim Q$ implies $P \subseteq Q$.

We work with the quotient set $PL^{\{\cdot,\_,\_^*\}}_{/\sim}$ rather than with $PL^{\{\cdot,\_,\_^*\}}$ directly: A word from $PL^{\{\cdot,\_,\_^*\}}$ is in *normal form* if it has no consecutive variable-length wildcards, i.e., if it has no consecutive "$\_^*$" and no occurrence of "$\_^*.\_$". Note that, each congruence class contains a unique word in normal form. Each word from $PL^{\{\cdot,\_,\_^*\}}$ can be transformed into normal form in linear time, just by removing superfluous variable-length wildcards and replacing each occurrence of "$\_^*.\_$" by "$\_.\_^*$". The *length* $|Q|$ of a $PL^{\{\cdot,\_,\_^*\}}$ expression Q is the number of labels in Q plus the number of wildcards (counting both variable-length and single-label wildcards) in the normal form of Q.

The empty path expression ε has length 0. The natural homomorphism from $PL^{\{\cdot,\_,\_^*\}}$ to $PL^{\{\cdot,\_,\_^*\}}_{/\sim}$ is an isomorphism when restricted to words in normal form. By abuse of notation we use the words from $PL^{\{\cdot,\_,\_^*\}}$ to denote their respective congruence class.

For nodes v and v' of an XML tree T, the *value intersection* of $v[\![Q]\!]$ and $v'[\![Q]\!]$ is given by $v[\![Q]\!] \cap_v v'[\![Q]\!] = \{(w, w') \mid w \in v[\![Q]\!], w' \in v'[\![Q]\!], w =_v w'\}$. That is, $v[\![Q]\!] \cap_v v'[\![Q]\!]$ consists of all those node pairs in T that are value equal and are reachable from v and v', respectively, by following Q-paths.

3 Keys for XML Data

The expressiveness of our fragment of XML keys results from the generality of our notion of value-equality and that of the path language. For more expressive path languages the containment problem becomes at least intractable [20].

Definition 1. *An XML key φ in is an expression of the form*

$$(Q_\varphi, (Q'_\varphi, \{P_1^\varphi, \ldots, P_{k_\varphi}^\varphi\}))$$

where Q_φ, Q'_φ, P_1^φ, …, $P_{k_\varphi}^\varphi \in PL^{\{\cdot,\_,\_^*\}}$ such that $Q_\varphi.Q'_\varphi.P_1^\varphi$, …, $Q_\varphi.Q'_\varphi.P_{k_\varphi}^\varphi$ are valid, and where k_φ is a non-negative integer.

An XML tree T satisfies φ if and only if $\forall q \in [\![Q_\varphi]\!] \, \forall q'_1, q'_2 \in q[\![Q'_\varphi]\!]$

$$\left(\bigwedge_{1 \le i \le k} q'_1[\![P_i^\varphi]\!] \cap_v q'_2[\![P_i^\varphi]\!] \neq \emptyset \right) \Rightarrow q'_1 = q'_2.$$

Herein, Q_φ is called the context path*, Q'_φ is called the* target path*, and $P_1^\varphi, \ldots, P_{k_\varphi}^\varphi$ are called the* key paths *of φ.*

By the previous definition, if the set of key paths is non empty (i.e. $k \ge 1$), a key φ is satisfied by a tree T if and only if for every node $q \in [\![Q_\varphi]\!]$ and all

nodes $q'_1, q'_2 \in q[\![Q'_\varphi]\!]$ such that there are nodes $x_i \in q'_1[\![P_i^\varphi]\!], y_i \in q'_2[\![P_i^\varphi]\!]$ with $x_i =_v y_i$ for all $i = 1, \ldots, k$, then $q'_1 = q'_2$. On the other hand, if the set of key path is empty, a key φ is satisfied by a tree T if and only if for every node $q \in [\![Q_\varphi]\!]$ there is at most one node reachable from q by following a Q'_φ-path, i.e., $q[\![Q'_\varphi]\!]$ contains at most one element. Thus, these latter keys identify nodes (within certain subtrees) by a unique path while the keys with nonempty sets of key paths identify nodes (within subtrees) by unique data values. Hence, we introduce the term *structural keys* for keys that have an empty set of key path expressions.

Example 4. We formalize the XML keys discussed in the introduction over XML trees with the layout of the tree T depicted in Figure 1.

a. $(\varepsilon, (bank, \{name\}))$ expresses "a *bank* node can be uniquely identified by the value of its *name* attribute". Over trees with this layout, the same can be expressed by $(\varepsilon, (\_, \{name\})$ and also by $(\varepsilon, (\_, \{\_^*.name\}))$ among others.
b. $(\_^*.account, (transaction, \{\_\}))$ expresses "in every subtree rooted at an *account* node, a *transaction* node can be uniquely identified by the value of its child node, which can either be a *withdrawal* node or a *deposit* node". As an alternative over trees with this layout, we could use $(\_.\_.\_.account, (transaction, \{\_\}))$ and $(bank.branch.client.\ account, (transaction, \{\_\}))$ among others.
c. $(\_^*.bank, (\_^*.client, \{cno, account\}))$ expresses "in every subtree rooted at a *bank* node, a *client* node can be uniquely identified by its child attribute *cno* together with its child node *account*". Over trees with this layout the same is expressed, among others, by $(\_, (\_.client, \{cno, account\}))$.
d. $(\_^*.transaction, (\_, \emptyset))$ expresses "Every *transaction* node has at most one child node".

3.1 Expressive and Tractable Fragments of XML Keys

In order to take advantage of XML keys effectively it becomes necessary to reason about them efficiently. Central to this task is the implication problem. Let $\Sigma \cup \{\varphi\}$ be a finite set of XML keys in a fragment $\mathcal{C}$. We say that Σ *(finitely) implies* φ, denoted by $\Sigma \models_{(f)} \varphi$, if and only if every (finite) XML tree T that satisfies all $\sigma \in \Sigma$ also satisfies φ. The *(finite) implication problem* for the fragment $\mathcal{C}$ is to decide, given any finite set of XML keys $\Sigma \cup \{\varphi\}$ in $\mathcal{C}$, whether $\Sigma \models_{(f)} \varphi$. If Σ is a finite set of XML keys in $\mathcal{C}$ let $\Sigma^*_{(f)}$ denote its *(finite) semantic closure*, i.e., the set of all XML keys (finitely) implied by Σ. That is, $\Sigma^*_{(f)} = \{\varphi \in \mathcal{C} \mid \Sigma \models_{(f)} \varphi\}$.

As shown in [7,8,15,16,11], efficient decision algorithm for the finite implication problem of XML keys exist for the following fragments.

$$\mathcal{K}_1 = \{(Q, (Q', \{P_1, \ldots, P_k\})) \mid k \geq 1, Q, Q' \in PL^{\{\cdot,\_^*\}} \text{ and } P_1, \ldots, P_k \in PL^{\{\cdot\}}\}$$
$$\mathcal{K}_2 = \{(Q, (Q', \{P_1, \ldots, P_k\})) \mid k \geq 0, Q, Q' \in PL^{\{\cdot,\_^*\}} \text{ and } P_1, \ldots, P_k \in PL^{\{\cdot\}}\}$$
$$\mathcal{K}_3 = \{(Q, (Q', \{P_1, \ldots, P_k\})) \mid k \geq 1, Q, Q' \in PL^{\{\cdot,\_,\_^*\}} \text{ and } P_1, \ldots, P_k \in PL^{\{\cdot,\_\}}\}$$

Note that the fragment $\mathcal{K}_2$ includes the fragment $\mathcal{K}_1$ plus structural XML keys with context and target paths in $PL^{\{\cdot,\_^*\}}$. The fragment $\mathcal{K}_3$ is defined using a more expressive path language than $\mathcal{K}_1$ which allows the specification of single-label wildcards. Thus, it also strictly includes the fragment $\mathcal{K}_1$. Regarding the relationship between $\mathcal{K}_2$ and $\mathcal{K}_3$, it is clear that there are XML keys in $\mathcal{K}_2$ which are not in $\mathcal{K}_3$ and vice versa. For instance, the key $(\varepsilon, (\_, \{name\}))$ belongs to $\mathcal{K}_3$ and does not belong to $\mathcal{K}_2$. On the contrary, the structural key $(bank.branch.client.account.transaction, (deposit, \emptyset))$ belongs to $\mathcal{K}_2$ and does not belong to $\mathcal{K}_3$.

In this work, we study a fragment of XML keys which strictly includes $\mathcal{K}_1$, $\mathcal{K}_2$ and $\mathcal{K}_3$. We define it as follows.

$$\text{Max-Keys} = \{\ (Q, (Q', \{P_1, \ldots, P_k\})) \mid k \geq 0, Q, Q', P_1, \ldots, P_k \in PL^{\{\cdot,\_,\_^*\}} \text{ but such that } Q' \text{ or } P_1, \ldots, P_k \in PL^{\{\cdot,\_\}}\}$$

It is easy to find examples which belong to Max-Keys and do not belong to any of the fragments $\mathcal{K}_1$, $\mathcal{K}_2$ and $\mathcal{K}_3$. Take for instance $(\varepsilon, (\_, \{\_^*.name\}))$ and $(\_^*.transaction, (\_, \emptyset))$ from Example 4.

The fragment Max-Keys of XML keys provides XML engineers with an enhanced ability to capture interesting properties of XML data. These are useful for several tasks in XML practice such as data integration, cleaning and validation among others. In the remainder of the article we will establish that despite its expressiveness, the fragment of Max-Keys can be reasoned about efficiently.

For the time being, we assume the number of key paths k to be ≥ 1 for all Max-Keys. We consider the case of Max-Keys with empty sets of key paths latter in Subsection 5.4.

The plan to verify the tractability of Max-Keys is as follows: First we will characterize the implication problem associated with Max-Keys in terms of a finite axiomatization. We can speak of *the* implication problem as the finite and unrestricted implication problems coincide for the fragment of Max-Keys. The axiomatization provides complete insight into the interaction of Max-Keys. This insight allows us to characterize the implication problem by constructive graph properties. This characterization enables us to establish a compact, low-degree polynomial-time algorithm for deciding implication.

4 Inference Rules for Max-Keys

Our goal is to establish a finite axiomatization for the implication of Max-Keys. To begin with we assemble a set of inference rules that allow us to derive new Max-Keys from given ones. Derivability with respect to a set $\mathfrak{R}$ of inference rules, denoted by the binary relation $\vdash_{\mathfrak{R}}$ between a set of Max-Keys and a single Max-Key, can be defined analogously to the notion in the relational data model [1, pp. 164-168].

We aim to find a set of inference rules which is *sound* and *complete* for the implication of Max-Keys. A set $\mathfrak{R}$ of inference rules is sound (respectively, complete) for the implication of Max-Keys if for all finite sets Σ of Max-Keys we

have $\Sigma^+_{\mathfrak{R}} \subseteq \Sigma^*$ (respectively, $\Sigma^* \subseteq \Sigma^+_{\mathfrak{R}}$). Herein, $\Sigma^+_{\mathfrak{R}} = \{\varphi \mid \Sigma \vdash_{\mathfrak{R}} \varphi\}$ denotes the *syntactic closure* of Σ under derivation by $\mathfrak{R}$.

Table 1 shows the set of inference rules for the implication of Max-Keys. Each inference rule has the form $\frac{premises}{conclusion}\, condition$ with premises from Max-Keys. That is, the path expressions used in the premises are always chosen such that the respective XML key lies in the fragment of Max-Keys.

Table 1. A Finite Axiomatization for Max-Keys

$$\frac{(Q,(Q',S))}{(Q,(Q',S\cup\{P\}))}\, Q' \text{ or } P \in PL^{\{\cdot,\_\}}$$
(prefix-epsilon)

$$\frac{}{(Q,(\epsilon,S))}$$
(epsilon)

$$\frac{(Q,(Q'.Q'',S))}{(Q.Q',(Q'',S))}$$
(target-to-context)

$$\frac{(Q,(Q',S\cup\{\epsilon,P\}))}{(Q,(Q',S\cup\{\epsilon,P.P'\}))}$$
(superkey)

$$\frac{(Q,(Q'.P,\{P'\}))}{(Q,(Q',\{P.P'\}))}\, \text{at least 2 of } Q', P, P' \in PL^{\{\cdot,\_\}}$$
(subnodes)

$$\frac{(Q,(Q',S))}{(Q'',(Q',S))}\, Q''\subseteq Q$$
(context-path-containment)

$$\frac{(Q,(Q'.P,\{\epsilon,P'\}))}{(Q,(Q',\{\epsilon,P.P'\}))}\, \text{at least 2 of } Q', P, P' \in PL^{\{\cdot,\_\}}$$
(subnodes-epsilon)

$$\frac{(Q,(Q',S))}{(Q,(Q'',S))}\, Q''\subseteq Q'$$
(target-path-containment)

$$\frac{(Q,(Q',\{P.P_1,\ldots,P.P_k\})),\ (Q.Q',(P,\{P_1,\ldots,P_k\}))}{(Q,(Q'.P,\{P_1,\ldots,P_k\}))}$$
(interaction)

$$\frac{(Q,(Q',S\cup\{P\}))}{(Q,(Q',S\cup\{P'\}))}\, P'\subseteq P$$
(key-path-containment)

We prove below the soundness of the *subnodes* rule since it provides valuable insight that explains out definition of Max-Keys. The soundness of the remaining rules can be shown using similar arguments. Therefore we omit those lengthy, but not very difficult proofs. For comparing, we also refer to our soundness proofs in [15,16,11] for the special cases of the smaller fragments $\mathcal{K}_1$, $\mathcal{K}_2$ and $\mathcal{K}_3$ of XML keys.

Lemma 1. *The* subnodes *rule is sound for the implication of XML keys in the fragment of Max-Keys.*

Proof. Suppose an XML tree T violates $(Q,(Q',\{P.P'\}))$. Then there is some node $q \in [\![Q]\!]$ and there are some nodes $q'_1, q'_2 \in q[\![Q']\!]$ such that $q'_1 \neq q'_2$ and such that there exist $p'_1 \in q'_1[\![P.P']\!]$ and $p'_2 \in q'_2[\![P.P']\!]$ where $p'_1 =_v p'_2$ holds. By definition of concatenation, there exist $p_1 \in q'_1[\![P]\!]$ and $p_2 \in q'_2[\![P]\!]$ such that $p'_1 \in p_1[\![P']\!]$ and $p'_2 \in p_2[\![P']\!]$ hold. Since T is a tree and due to the condition of the *subnodes* rule Q' or $P.P'$ is a $PL^{\{\cdot,\_\}}$ expression (i.e. has no variable-length wildcards), it holds that q'_1 is neither an ancestor nor a descendant of q'_2.

Consequently, $p_1 \neq p_2$ (since otherwise $q'_1 = q'_2$). This, however, means that T also violates $(Q, (Q'.P, \{P\}))$. □

In the proof of the previous lemma we have made an interesting observation: If we allow the use of variable-length wildcards in the target and key paths at the same time, then we could have a pair of distinct target nodes q'_1 and q'_2 so that one is an ancestor of the other one. This would allow us to build a counter-example tree to demonstrate that the *subnodes* rule is not sound for such an extended fragment of Max-Keys.

5 Axiomatic Characterization of Max-Keys Implication

Our goal is to demonstrate that the set $\mathfrak{R}$ of inference rules is also complete for the implication of Max-Keys. Completeness means we need to show that for an arbitrary finite set $\Sigma \cup \{\varphi\}$ of Max-Keys with $\varphi \notin \Sigma^+_{\mathfrak{R}}$ there is some XML tree T that satisfies all members of Σ but violates φ. That is, T is a counter-example tree for the implication of φ by Σ.

The general proof strategy is as follows: For T to be a counter-example we i) require a context node q_φ with (at least) two target nodes q'_φ that have value equal key paths $q^\varphi_1, \ldots, q^\varphi_{k_\varphi}$, and ii) must for every context node q_σ not have target nodes q'_σ with value equal key nodes $q^\sigma_1, \ldots, q^\sigma_{k_\sigma}$, for each $\sigma \in \Sigma$. Such a counter-example tree exists if and only if these two conditions can be satisfied simultaneously.

In a fist step, we represent φ as a finite node-labeled tree $T_{\Sigma,\varphi}$, which we call the *mini-tree*. Then, we reverse the edges of the mini-tree and add to the resulting tree downward edges for certain members of Σ. This final digraph $G_{\Sigma,\varphi}$ is called the *witness network*. A downward edge resulting from σ tells us that under each source node there can be at most one target node. Now, if we can reach the target node of φ from its context node along a dipath then there is no counter-example tree T. In other words, if we satisfy condition ii) above, then we cannot satisfy condition i). Otherwise, we can construct a counter-example tree T.

5.1 Witness Networks

Let $\Sigma \cup \{\varphi\}$ be a finite set of Max-Keys. Let $\mathcal{L}_{\Sigma,\varphi}$ denote the set of all labels $\ell \in \mathcal{L}$ that occur in path expressions of members in $\Sigma \cup \{\varphi\}$, and fix a label $\ell_0 \in \mathbf{E} - \mathcal{L}_{\Sigma,\varphi}$. First we transform the path expressions occurring in φ into simple path expressions in $PL^{\{\cdot\}}$. For that purpose we replace each single-label wildcard "_" by ℓ_0 and each variable-length wildcard "_*" by a sequence of $l+1$ labels ℓ_0, where l is the maximum number of consecutive single-label wildcards that occurs in any Max-Key in $\Sigma \cup \{\varphi\}$. This transformation turns Q_φ into O_φ, Q'_φ into O'_φ, and each P^φ_i into O^φ_i for $i = 1, \ldots, k_\varphi$. The path expressions after the transformation do not contain any more wildcards (neither single-label nor variable-length ones).

The proper choice of the integer l is essential for the later construction. In particular, if there are no occurrences of single-label wildcards in the Max-Keys under consideration, then $l = 0$ and we just replace each variable-length wildcard "$\_^*$" by one ℓ_0.

To continue with our construction, let p be an O_φ-path from a node r_φ to a node q_φ, let p' be an O'_φ-path from a node r'_φ to a node q'_φ and, for $i = 1, \ldots, k_\varphi$, let p_i be a O_i^φ-path from a node r_i^φ to a node x_i^φ, such that the paths $p, p', p_1, \ldots, p_{k_\varphi}$ are mutually node-disjoint. From the paths $p, p', p_1, \ldots, p_{k_\varphi}$ we obtain the *mini-tree* $T_{\Sigma,\varphi}$ by identifying the node r'_φ with q_φ, and by identifying each of the nodes r_i^φ with q'_φ.

The *marking* of the mini-tree $T_{\Sigma,\varphi}$ is a subset $\mathcal{M}$ of the node set of $T_{\Sigma,\varphi}$: if for all $i = 1, \ldots, k_\varphi$ we have $Q_i^\varphi \neq \varepsilon$, then $\mathcal{M}$ consists of the leaves of $T_{\Sigma,\varphi}$, and otherwise $\mathcal{M}$ consists of all descendant nodes of q'_φ in $T_{\Sigma,\varphi}$.

We use mini-trees to calculate the impact of Max-Keys in Σ on a possible counter-example tree for the implication of φ by Σ. To distinguish Max-Keys that have an impact from those that do not, we introduce the notion of *applicability*. Intuitively, when a Max-Key is not applicable, then we do not need to satisfy it in a counter-example tree as it does not require all its key paths. Let $T_{\Sigma,\varphi}$ be the mini-tree of the Max-Key φ with respect to Σ, and let $\mathcal{M}$ be its marking. A Max-Key σ is said to be *applicable* to φ if there are nodes $w_\sigma \in [\![Q_\sigma]\!]$ and $w'_\sigma \in w_\sigma[\![Q'_\sigma]\!]$ in $T_{\Sigma,\varphi}$ such that $w'_\sigma[\![P_i^\sigma]\!] \cap \mathcal{M} \neq \emptyset$ for all $i = 1, \ldots, k_\sigma$. We say that w_σ and w'_σ *witness* the applicability of σ to φ.

We define the *witness network* $G_{\Sigma,\varphi}$ of φ and Σ as the node-labeled digraph obtained from $T_{\Sigma,\varphi}$ as follows: the nodes and node-labels of $G_{\Sigma,\varphi}$ are exactly the nodes and node-labels of $T_{\Sigma,\varphi}$, respectively. The edges of $G_{\Sigma,\varphi}$ consist of the reversed edges from $T_{\Sigma,\varphi}$. Furthermore, for each Max-Key $\sigma \in \Sigma$ that is applicable to φ and for each pair of nodes $w_\sigma \in [\![Q_\sigma]\!]$ and $w'_\sigma \in w_\sigma[\![Q'_\sigma]\!]$ that witness the applicability of σ to φ we add a directed edge (w_σ, w'_σ) to $G_{\Sigma,\varphi}$. We refer to these additional edges as *witness edges* while the reversed edges from $T_{\Sigma,\varphi}$ are referred to as *upward edges* of $G_{\Sigma,\varphi}$. This is the case since for every witness w_σ and w'_σ the node w'_σ is a descendant of the node w_σ in $T_{\Sigma,\varphi}$, and thus the witness edge (w_σ, w'_σ) is a downward edge or loop in $G_{\Sigma,\varphi}$.

Example 5. Let us consider the following Max-Keys which were specified by the database designer for XML trees with the layout of the tree in Figure 1.

$$\begin{aligned}
\varphi &= (\varepsilon, (bank._^*.client.account, \{no, kind\})) \\
\sigma_1 &= (\varepsilon, (bank._, \{_^*.account.no\})) \\
\sigma_2 &= (bank, (_^*.client, \{_no\})) \\
\sigma_3 &= (_^*.client, (account, \{kind\}))
\end{aligned}$$

Let $\Sigma = \{\sigma_1, \sigma_2, \sigma_3\}$. The left picture of Figure 3 shows the mini-tree $T_{\Sigma,\varphi}$ and its marking (note that leaves are marked by $\times$). The right picture in the same figure shows its corresponding witness network $G_{\Sigma,\varphi}$. Note that all the Max-Keys in Σ are applicable to φ. On the other hand, the Max-Key $\sigma_4 = (\varepsilon, (bank.branch.client.account, \{no, kind\}))$ is not applicable to φ. In fact, σ_4

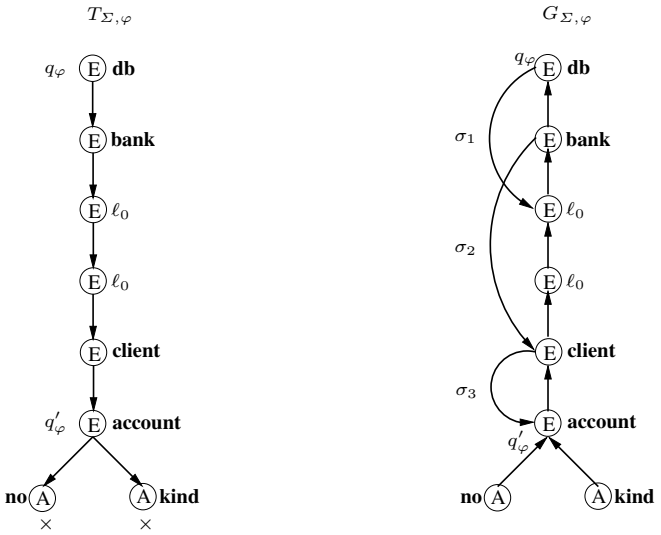


Fig. 3. A mini-tree and a witness network

is implied by φ since it can be derived by an application of the *target-path-containment* rule.

In the following section we will prove the following crucial fact. If q'_φ is reachable from q_φ in $G_{\Sigma,\varphi}$, then $\varphi \in \Sigma^+_{\mathfrak{R}}$. In other words, if φ is not derivable from Σ, then there is *no* dipath from q_φ to q'_φ in $G_{\Sigma,\varphi}$.

5.2 Reachability Implies Derivability

The following important lemma holds the key to prove the completeness of $\mathfrak{R}$ and also for our low-degree polynomial time decision algorithm for the implication problem of Max-Keys.

Lemma 2. *Let $\Sigma \cup \{\varphi\}$ be a finite set of Max-Keys. If q'_φ is reachable from q_φ in the witness network $G_{\Sigma,\varphi}$, then $\varphi \in \Sigma^+_{\mathfrak{R}}$.*

Proof (Sketch). Let D denote the simple path in $G_{\Sigma,\varphi}$ from q_φ to q'_φ. According to the definition of the witness network we can assume without loss of generality that D consists of a sequence $\pi_1, \ldots, \pi_{n+1}$, $n \geq 1$, where for each $i = 1, \ldots, n$, π_i starts with a possibly empty sequence of upward edges followed by a single witness edge $(w_{\sigma_i}, w'_{\sigma_i})$ where w_{σ_i} and w'_{σ_i} witness the applicability of σ_i to φ, and π_{n+1} is a possibly empty sequence of upward edges. Moreover, we can assume that $q_\varphi, w'_{\sigma_1}, \ldots, w'_{\sigma_n}$ form a proper descendant chain, q'_φ is a proper descendant of $w'_{\sigma_{n-1}}$ and w'_{σ_n} is a descendant node of q'_φ in $T_{\Sigma,\varphi}$. This situation is illustrated by the witness network $G_{\Sigma,\varphi}$ in Figure 3.

We now describe a series of assumptions which we use to show that the existence of the witness edges in D implies the existence of a witness edge (q_φ, q'_φ) which results from a Max-Key σ in Σ^+.

1. The final witness edge in D can be replaced by a witness edge that ends in q'_φ. That is, we can assume without loss of generality that π_{n+1} is indeed an empty sequence and $w'_{\sigma_n} = q'_\varphi$ where the set of key paths of σ_n is $\{P_1^\varphi, \ldots, P_{k_\varphi}^\varphi\}$.
2. If there is a witness edge (w_σ, w'_σ) in the witness network $G_{\Sigma,\varphi}$ that corresponds to the applicability of some $\sigma \in \Sigma^+_{\mathfrak{R}}$ to φ, then for each node w between w_σ and w'_σ in $T_{\Sigma,\varphi}$ there is also a witness edge (w, w'_σ) in $G_{\Sigma,\varphi}$ which corresponds to the applicability of some $\sigma' \in \Sigma^+_{\mathfrak{R}}$ to φ. In our example in Figure 3, this indicates the existence of a witness edge from the first ℓ_0 node (from top to bottom) to the *client* node.
3. If there is a witness edge $(w_{\sigma_1}, w'_{\sigma_1})$ and another witness edge $(w'_{\sigma_1}, q'_\varphi)$, then there is also a witness edge $(w_{\sigma_1}, q'_\varphi)$. In our example in Figure 3, this indicates the existence of a witness edge from the *db* node to the *account* node.

The strategy to prove the previous assumptions consists on showing that for each new witness edge D' obtained by virtue of the assumptions, there is a corresponding inference by $\mathfrak{R}$ of a Max-Key σ' that corresponds to the witness edge D'. For the sake of presentation, we omit this lengthy but not very difficult part of the proof.

From Assumption 1 and 2, we can conclude that there is a simple path D' in $G_{\Sigma,\varphi}$ from q_φ to q'_φ. In fact, D' consists of the sequence $\pi'_1, \ldots, \pi'_n$ where each π'_i with $1 \leq i \leq n$ consists of a single witness edge $(w_{\sigma_i}, w'_{\sigma_i})$ where $w'_{\sigma_i} = w_{\sigma_{i+1}}$ for $i = 1, \ldots, n-1$ and where $w_{\sigma_1} = q_\varphi$ and $w'_{\sigma_n} = q'_\varphi$. Again, $q_\varphi, w'_{\sigma_1}, \ldots, w'_{\sigma_n}$ form a proper descendant chain.

At this stage we can use Assumption 3 repeatedly to conclude that there is a single witness edge $D_0 = (q_\varphi, q'_\varphi)$ in $G_{\Sigma,\varphi}$ resulting from the Max-Key $\sigma = (Q_\sigma, (Q'_\sigma, \{P_1^\varphi, \ldots, P_{k_\varphi}^\varphi\}))$ in $\Sigma^+_{\mathfrak{R}}$ that is applicable to φ. Due to the applicability of σ to φ we conclude that $Q_\varphi \subseteq Q_\sigma$ and $Q'_\varphi \subseteq Q'_\sigma$. We can now apply the *context-path-containment* and *target-path-containment* rule to obtain $(Q_\varphi, (Q'_\varphi, \{P_1^\varphi, \ldots, P_{k_\varphi}^\varphi\})) \in \Sigma^+_{\mathfrak{R}}$, which proves the lemma. □

The next example illustrates how the edges of the witness network encode an inference by $\mathfrak{R}$.

Example 6. Let φ and Σ be as in Example 5. Recall that the right picture of Figure 3 shows the corresponding witness network. Thus by Lemma 2, $\Sigma \vdash_{\mathfrak{R}} \varphi$. Next, we show the actual derivation by $\mathfrak{R}$. We first apply the *superkey* rule to σ_3 to derive $\sigma'_3 = (\_^*.client, (account, \{no, kind\}))$ (cf. with Assumption 1 in the proof of Lemma 2). We also apply the *superkey* rule to σ_1 and σ_2 to derive $\sigma'_1 = (\varepsilon, (bank.\_, \{\_^*.account.no, \_^*.account.kind\}))$ and $\sigma'_2 = (bank, (\_^*.client, \{\_.no, \_.kind\}))$, respectively. Then we apply the *target-to-context* rule to σ'_2 to derive $\sigma''_2 = (bank.\_^*, (client, \{\_.no, \_.kind\}))$ (cf. with Assumption 2 in the proof

of Lemma 2). It is easy to see that by successively applying the *context-path-containment*, *target-path-containment* and *key-path-containment* rules, we can derive from σ_1', σ_2'' and σ_3' the following Max-Keys, respectively.

$$\alpha_1 = (\varepsilon, (bank.\_^*, \{client.account.no, client.account.kind\}))$$
$$\alpha_2 = (bank.\_^*, (client, \{account.no, account.kind\}))$$
$$\alpha_3 = (bank.\_^*.client, (account, \{no, kind\}))$$

Finally, by application of the *interaction* rule to α_2 and α_3 we obtain $\alpha_4 = (bank.\_^*, (client.account, \{no, kind\}))$, and again by application of the *interaction* rule to α_1 and α_4 we derive φ (cf. with Assumption 3 in the proof of Lemma 2).

5.3 Completeness

We have now the tools to prove the completeness of our set of inference rules.

Theorem 1. *The inference rules in Table 1 are complete for the implication of Max-Keys.*

Proof (Sketch). Let $\Sigma \cup \{\varphi\}$ be a finite set of Max-Keys such that $\varphi \notin \Sigma^+_{\mathfrak{R}}$. We will show that $\varphi \notin \Sigma^*$ by constructing a finite XML tree T which satisfies all Max-Keys in Σ but does not satisfy φ. Since $\varphi \notin \Sigma^+_{\mathfrak{R}}$ we know by Lemma 2 that there is no path from q_φ to q'_φ in the witness network $G_{\Sigma,\varphi}$. Let u denote the bottom-most descendant node of q_φ in $T_{\Sigma,\varphi}$ that is reachable from q_φ in $G_{\Sigma,\varphi}$. Note that u is a proper ancestor of q'_φ in $T_{\Sigma,\varphi}$ since otherwise u and thus q'_φ were reachable from q_φ in $G_{\Sigma,\varphi}$.

Let T_0 denote a copy of the path from r to u in $T_{\Sigma,\varphi}$, and T_1, T_2 denote two node-disjoint copies of the subtree of $T_{\Sigma,\varphi}$ rooted at u. We want that a node of T_1 and a node of T_2 become value equal precisely when they are copies of the same marked node in $T_{\Sigma,\varphi}$. For attribute and text nodes this is achieved by choosing string values accordingly, while for element nodes we can adjoin a new child node with a label from $\mathcal{L} - (\mathcal{L}_{\Sigma,\varphi} \cup \{\ell_0\})$ to achieve this. The counter-example tree T is obtained from T_0, T_1, T_2 by identifying the leaf node u of T_0 with the root nodes of T_1 and T_2. We conclude that T violates φ, and our construction guarantees that T satisfies all $\sigma \in \Sigma$. □

The construction of the counter example tree T in the proof of Theorem 1 is illustrated by the following example.

Example 7. Let σ_1, σ_2 and φ be as in Example 5. The corresponding mini-tree $T_{\Sigma,\varphi}$, witness network $G_{\Sigma,\varphi}$ and counter-example tree T for the implication of φ by $\Sigma = \{\sigma_1, \sigma_2\}$ are shown in Figure 4. Note that T satisfies σ_1 and σ_2, and violates φ.

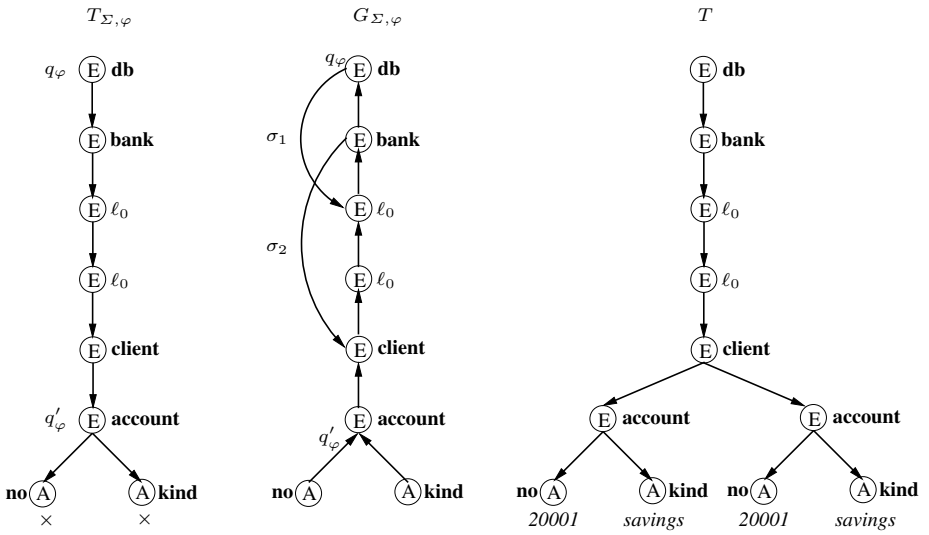


Fig. 4. Mini-tree, witness network and counter-example tree

5.4 Dealing with Structural Max-Keys

Max-Keys with empty sets of key paths behave quite differently than Max-Keys with non empty key paths. The former ones identify nodes (within certain subtrees) by a unique context path. The latter ones identify nodes (within subtrees) by unique data values.

Let us consider Max-Keys of the form $\alpha = (Q, (Q', \emptyset))$ whose key path Q' may contain single-label wildcards, but no variable-length wildcards. It is not hard to see that the inference rules in Table 1 still hold for this case. Using the *superkey* rule we can derive $\alpha' = (Q, (Q', \{\_^*\}))$ from α.

Thus when constructing the witness network $G_{\Sigma,\varphi}$ for some Σ and φ in Max-Keys such that α belongs to Σ then we will replace α by α'. As we have demonstrated above the counter-example tree T constructed for φ and the resultant Σ' would satisfy Σ' but violate φ. However, it is easy to validate that by our construction T satisfies not only α' but even α. So T would actually satisfy Σ but violate φ, thus showing that Σ does not imply φ.

If φ has an empty set of key paths, then we will construct the witness network $G_{\Sigma,\varphi'}$ where $\varphi' = (Q_\varphi, (Q'_\varphi, \{\_^*\}))$. As we have demonstrated above the counter-example tree T constructed for φ' and Σ would satisfy Σ but violate φ'. Consequently, it would also violate φ (as φ implies φ'), thus showing that Σ does not imply φ.

Finally, note that $\varphi' = (Q_\varphi, (Q'_\varphi, \{\_^*\}))$ does not generally imply φ. A counter-example tree T can be easily constructed from merging two copies of $T_{\Sigma,\varphi'}$ on all nodes other than $q'_{\varphi'}$ and then assigning two different string values to the two copies of $q'_{\varphi'}$. Then T would satisfy φ' but violate φ.

6 A Decision Algorithm for Max-Keys Implication

We will now design a low-degree polynomial time decision algorithm for the implication problem of Max-Keys. It is based on the following characterization of the implication problem in terms of the reachability problem of nodes in the witness network.

Theorem 2. *Let $\Sigma \cup \{\varphi\}$ be a finite set of Max-Keys. We have $\Sigma \models \varphi$ if and only if q'_φ is reachable from q_φ in the witness network $G_{\Sigma,\varphi}$.*

Proof. Assume that q'_φ is not reachable from q_φ in $G_{\Sigma,\varphi}$. Then we can generate a counter-example tree for the implication of φ by Σ as described in the proof of Theorem 1. It follows that Σ does not imply φ.

On the contrary, assume now that q'_φ is indeed reachable from q_φ in $G_{\Sigma,\varphi}$. We conclude by Lemma 2 that φ is implied by Σ. □

Theorem 2 tells us that we can decide implication by constructing the witness network and testing reachability in $G_{\Sigma,\varphi}$ by applying well known search techniques. This establishes a surprisingly compact algorithm.

Algorithm 1. Decision Algorithm for the Implication Problem of Max-Keys

Input: finite set of Max-Keys $\Sigma \cup \{\varphi\}$
Output: yes, if $\Sigma \models \varphi$; no, otherwise
1: Construct $G_{\Sigma,\varphi}$ for Σ and φ;
2: **if** q'_φ is reachable from q_φ in $G_{\Sigma,\varphi}$ **then return** yes;
else return no; **end if**

The presentation of Algorithm 1 to decide the implication of Max-Keys remains the same as Algorithm 4.2 in [15] to decide the implication of the strictly less expressive fragment $\mathcal{K}_1$ of XML keys. However, the construction of the witness network $G_{\Sigma,\varphi}$, which is central to both algorithms, requires considerably more effort for the more expressive fragment of Max-Keys. This effort results in an slight increase in the worst-case time complexity of the algorithm. Nevertheless, the simplicity of Algorithm 1 enables us to conclude that the implication of Max-Keys can be decided in low-degree polynomial time in the worst case.

6.1 Implementation and Complexity Analysis of Algorithm 1

In this subsection we discuss our implementation of Algorithm 1 and analyze its theoretical complexity.

Data Structures. We need data structures suitable to represent mini-trees and witness networks. The obvious candidates are adjacency matrices and adjacency lists [2]. Since the algorithm does not require frequent determination of edge existence, we choose the latter in order to minimize the memory requirements.

In our implementation, a mini-tree $T_{\Sigma,\varphi}$ is represented by using a list L of length $n = |V|$ where V is the vertex set of $T_{\Sigma,\varphi}$. Each element $e_i \in L$ is represented by an object of type *vertexEle* that has a pointer to the adjacency list of the i-th vertex v_i in some fixed enumeration of the vertices in V, a pointer to the data component of the vertex v_i, and a pointer to the next element e_{i+1} in the list. In turn, the data component of a vertex v_i is represented by an object of type *nodeEle*, and an element in the adjacency list of a vertex v_i is represented by an object of type *edgeEle*. An object of type *nodeEle* has an *id* component that uniquely identifies v_i, a *label* component with the label of v_i, a flag *visited*, and a *type* component with the type E (element), A (attribute) or S (PCDATA) of v_i. An object of type *edgeEle* has a pointer to an object of type *vertexEle* and a pointer to the next object of type *edgeEle* in the adjacency list. Witness networks are represented likewise.

Implementation. We implemented Step 1 of Algorithm 1, using the following strategy:

i. Construct $T_{\Sigma,\varphi}$;
ii. Determine the marking of $T_{\Sigma,\varphi}$;
iii. Reverse all edges of $T_{\Sigma,\varphi}$;
iv. For each $\sigma \in \Sigma$, add the edge (w_σ, w'_σ) to $T_{\Sigma,\varphi}$ whenever w_σ and w'_σ witness the applicability of σ to φ.

Substep (i) involves constructing the mini-tree $T_{\Sigma,\varphi}$ using the data structures defined at the beginning of this section. Note that we can find a label ℓ_0 that is not among the labels used in the XML keys in $\Sigma \cup \{\varphi\}$ in time $\sum_{\sigma_i \in \Sigma} |\sigma_i| + |\varphi|$, where $|\sigma_i|$ and $|\varphi|$ denote the sum of the lengths of all path expressions in σ_i and φ, respectively. Once we have got a suitable label, ℓ_0, $T_{\Sigma,\varphi}$ can be built in time $\mathcal{O}(|\varphi| \times l)$, where $|\varphi|$ is the sum of the lengths of all path expressions in φ and l is the maximum number of consecutive single-label wildcards that occurs in Σ. Note that the mini-tree $T_{\Sigma,\varphi}$ has at most $(|\varphi| \times l) + 1$ nodes.

Regarding Substep (ii), if $P_i^\varphi \neq \varepsilon$ we can determine the marking of the mini-tree $T_{\Sigma,\varphi}$ by simply traversing the list L marking the nodes whose adjacency list is empty. Note that those nodes correspond to leaves in $T_{\Sigma,\varphi}$. Otherwise, we mark all nodes in the adjacency list of the element e_i in L that represents q'_φ, and recursively mark all descendants of those nodes. This step takes $\mathcal{O}(|\varphi| \times l)$ time.

Assuming that the adjacency list of a vertex v_i lists the vertices v_j such that there is an edge from v_i to v_j. Substep (iii) involves the generation of a new adjacency list which corresponds to $T_{\Sigma,\sigma}$ with all its directed edges reverted. Again, this takes time $\mathcal{O}(|\varphi| \times l)$.

Substep (iv) requires, for each $\sigma \in \Sigma$, to evaluate $w'_\sigma[\![P_i^\sigma]\!]$ for $i = 1, \ldots, k_\sigma$, for all $w'_\sigma \in w_\sigma[\![Q'_\sigma]\!]$ and all $w_\sigma \in [\![Q_\sigma]\!]$. Note that a query of the form $v[\![Q]\!]$ is a Core XPath query and can be evaluated on a node-labeled three T in order $\mathcal{O}(|T| \times |Q|)$ time [13]. Hence, we can evaluate $w'_\sigma[\![P_i^\sigma]\!]$ for all $i = 1, \ldots, k_\sigma$ in time $\mathcal{O}(|\varphi| \times l \times |\sigma|)$. Since $[\![Q_\sigma]\!]$ and $w_\sigma[\![Q'_\sigma]\!]$ contain at most $|\varphi| \times l$ nodes each,

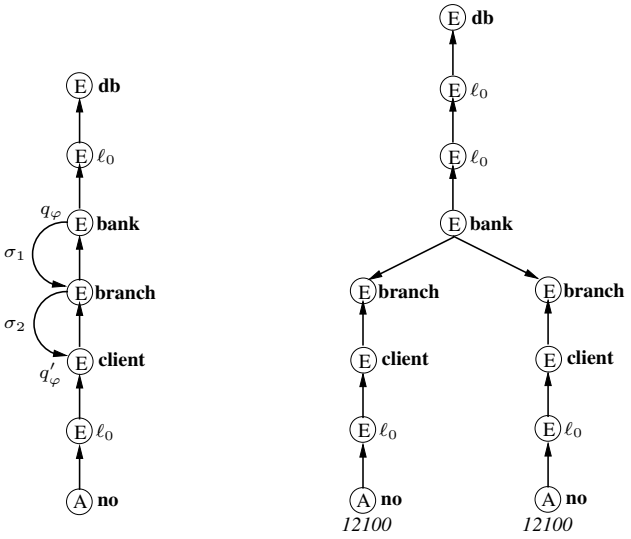


Fig. 5. Incorrectly built witness network and counter-example

this step can be executed in $\mathcal{O}((|\varphi| \times l)^3 \times |\sigma|)$ time for each σ. Hence, we need $\mathcal{O}(||\Sigma|| \times (|\varphi| \times l)^3)$ time to generate $G_{\Sigma,\varphi}$, where $||\Sigma||$ denotes the sum of all sizes $|\sigma|$ for $\sigma \in \Sigma$.

Finally, Step 2 of Algorithm 1 can be implemented by applying a depth-first search algorithm to $G_{\Sigma,\varphi}$ with root q_φ. This algorithm works in time linear in the number of edges of $G_{\Sigma,\varphi}$ [17]. Thus reachability can be decided in $\mathcal{O}(|\varphi+l|^2)$.

Complexity. The following result is a consequence of the previous observations.

Theorem 3. *If $\Sigma \cup \{\varphi\}$ is a finite set of Max-Keys, then the implication problem $\Sigma \models \varphi$ can be decided in $\mathcal{O}(||\Sigma|| \times (|\varphi| \times l)^3)$ time, where $|\varphi|$ is the sum of the lengths of all path expressions in φ, $||\Sigma||$ denotes the sum of all sizes $|\sigma|$ for $\sigma \in \Sigma$, l is the maximum number of consecutive single-label wildcards that occur in Σ.*

It is important to note the blow-up in the size of the counter-example with respect to φ. This is due to the occurrence of consecutive single-label wildcards. If the number l is fixed in advance, then Algorithm 1 establishes a worst-case time complexity that is quadratic in the input. In particular, if the input consists of XML keys in $\mathcal{K}_1$, as studied in [15,12], then the worst-case time complexity of Algorithm 1 is that of the algorithm dedicated to XML keys in $\mathcal{K}_1$ only [15].

Remark 1. If we simply replace each variable-length wildcard "_*" by the single-label ℓ_0 and not by a sequence of $l+1$ labels ℓ_0, then Theorem 2 does not hold. To see this, consider

$$\varphi = (\_^*.bank, (branch.client, \{\_no\}))$$
$$\sigma_1 = (\_.bank, (branch, \{client.\_no\}))$$
$$\sigma_2 = (\_^*.bank.branch, (client, \{\_no\}))$$

Let $\Sigma = \{\sigma_1, \sigma_2\}$. A simple replacement of "$\_^*$" by ℓ_0 results in the witness network shown on the first (from the left) picture in Figure 5. But then by Lemma 2, Σ would imply φ, which is clearly incorrect as shown by the counter-example tree on the second picture in Figure 5.

7 Applications and Performance Evaluation

In this section we present the results regarding the application and performance evaluation of our decision algorithm for the finite implication problem of Max-Keys. We also compare these results against the results obtained in [12] for a strictly less expressive fragment of XML keys. We start by describing in Subsection 7.1 the construction of the sets of XML keys used in our experiments. These set of keys correspond to actual XML documents which are also described in this section, and latter used in the experiments regarding the application of XML key reasoning in the context of XML document validation. These latter experiments are reported in Subsection 7.3. The experiments regarding the performance of the decision algorithm for the implication of Max-Keys are presented in Subsection 7.2.

The running times reported in Subsection 7.2 were obtained in a fairly standard Intel Core i7 2.8 GHz machine, with 4 GB of RAM, running a Linux kernel 2.6.32. We compiled our C++ implementation of the algorithms using the standard g++ compiler from the GNU Compiler Collection 4.6.3. The experiments in Subsection 7.3 were run on a cluster of 160 nodes of type Xeon E5-2680(i7) with 128 GB and 16 cores per node (2.68 GHZ) running Linux RHEL 6.1. The reason for using this facility was to save time on the experiments concerning the validation of XML documents, in particular this was required for the validation of the big XML document of 3.2GB corresponding to the Chilean electoral role. The use of these nodes allowed us to load in RAM the whole DOM representation of each XML tree tested in the experiments (including the one corresponding to the Chilean electoral role), thus simplifying the task of writing a validation algorithm to test our ideas. It should be noted that this task is accessory to this paper as it was only used to showcase the benefits of using XML key reasoning in this context. In this work, we are not concerned with parallelization challenges. In fact, we ran these jobs with a standard sequential algorithm, having one instance of an XML document and corresponding set of keys to be validated per node. We simply run several experiments (over different documents) at once by submitting different jobs (i.e., by using a separated node for each validation instance).

We remind the reader that, as stated in the introduction, all the data sets used in our experiments as well as the full set of results and the binary codes of the implemented algorithms are publicly available at `http://emir-munoz.github.com/xml-constraints`. To the best of our knowledge, this is the first

time that large sets of non-trivial XML keys for real XML documents are made publicly available. Our hope is that this contribution facilitates further research in the area.

7.1 Defining the Data Sets: XML Keys and XML Documents

We use a collection of XML documents from [22] plus a large XML document (*padron.xml*) that holds the publicly available Chilean electoral roll[1]. A characterization of these XML documents is shown in Table 2. From [22] we use the documents *321gone.xml* and *yahoo.xml* (auction data), *dblp.xml* (bibliographic information on CS), *nasa.xml* (astronomical data), *SigmodRecord.xml* (articles from SIGMOD Record), and *mondial-3.0.xml* (world geographic database).

Table 2. XML Documents

Doc ID	Document	No. of Elements	No. of Attributes	Size	Max. Depth	Average Depth
doc1	padron.xml	119,235,504	39,745,398	3.2 GB	5	4.06667
doc2	321gone.xml	311	0	24 KB	5	3.76527
doc3	yahoo.xml	342	0	25 KB	5	3.76608
doc4	dblp.xml	29,494	3,247	133.9 MB	6	2.90228
doc5	nasa.xml	476,646	56,317	25 MB	8	5.58314
doc6	SigmodRecord.xml	11,526	3,737	478 KB	6	5.14107
doc7	mondial-3.0.xml	22,423	47,423	1.9 MB	5	3.59274

In order to generate realistic XML keys for the experiments regarding the performance of the decision algorithm in Subsection 7.2 as well as to use the same sets of XML keys for the experiments involving documents validation in Subsection 7.3, we follow a strategy that is grounded in the XML documents described above. We explain this strategy using the XML document with the Chilean electoral roll as a model. The layout of this document is illustrated in Figure 6. For clarity of presentation we have translated the labels of the nodes from Spanish to English and simplified the structure by collapsing the first name, father's surname and mother's surname descendant nodes of the *name* element node into just one text node. We start by defining a series of keys which are appropriate in the context of this XML document. We list some of them together with an informal interpretation for reference.

a. $(\varepsilon, (commune, \{person\}))$
"A person cannot be enrolled in more than one commune".
b. $(\varepsilon, (commune, \{person.id\}))$
"Similar to (a), but not equivalent".

[1] The Chilean electoral roll can be downloaded in PDF format from the official site `http://www.servel.cl` of the Electoral Commission of Chile. The XML version used in this paper was extracted from the PDF document by Cristian Bravo-Lillo and can be obtained from `http://manzanamecanica.org/2012/11/padron_electoral_en_xml.html`

c. $(commune, (person, \{id\}))$
"A *person* node can be identified by its *id* attribute respectively to a *commune* node".

d. $(commune.person, (polling, \{circumscription, district\}))$
"A *polling* node can be identified by its child nodes *circumscription* and *district* node respectively to a *person* node".

e. $(commune.person, (polling, \emptyset))$
"A *person* node can only have one child node with label *polling*".

f. $(commune.person.polling, (circumscription, \emptyset))$
"A *polling* node can only have one child node with label *circumscription*".

g. $(\_.\_.\_, (circumscription, \emptyset))$
"Idem (f) over trees with the layout of the tree in Figure 6".

h. $(\varepsilon, (\_, \{\_^*.id\}))$
"Idem (b) over trees with the layout of the tree in Figure 6".

i. $(\_.\_, (\_^*.polling, \{\_S\}))$
"A *polling* node can be identified by the value of its descendant text nodes relatively to the nodes at level two in the XML tree".

j. $(\_.\_, (\_, \{\_^*.S\}))$ "Every node at level three can be identified by its descendant text nodes respectively to a node at level two. In trees with the layout of the tree in Figure 6, this means that for instance the name of a person cannot coincide with the address which in turn cannot coincide with the circumscription nor with the district".

Note that the XML keys in (a)–(d) are in the strictly less expressive fragment $\mathcal{K}_1$ of XML keys studied in [12] while the remaining keys are not. All the XML keys in the previous list are however in the fragment of Max-Keys covered in this work. Also note that the XML keys in (e)–(g) are structural keys and that, over trees with the layout of the tree in Figure 6, the XML keys in (i) and (j) cannot be expressed without allowing wildcards in the key paths.

We then define new (implied) XML keys, by successively applying the inference rules for Max-Keys from Table 1 to the previously defined set of XML keys.

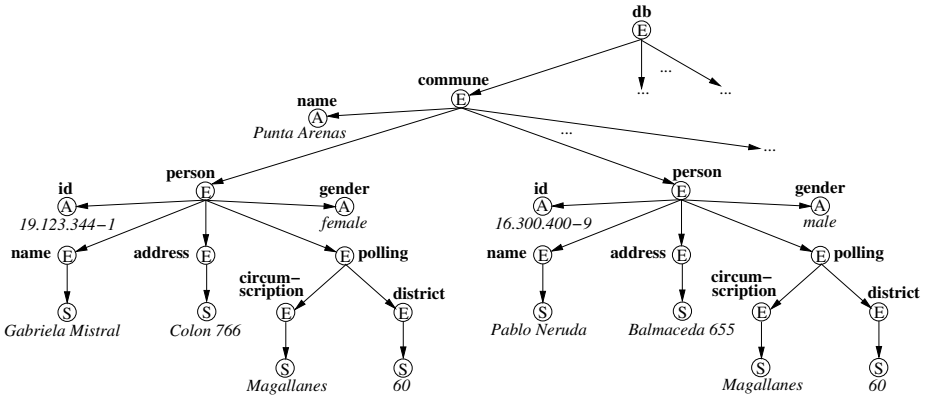


Fig. 6. Chilean electoral roll

For instance, by applying the *superkey* rule to the XML key in (h) we can obtain the new XML keys $(\varepsilon, (\_, \{\_^*.id, \_^*.gender\}))$, $(e, (\_, \{\_^*.id, \_^*.name\}))$ and $(e, (\_, \{\_^*.id, \_^*.gender, \_^*.name\}))$ among others.

Finally, we define sets $\Sigma \cup \{\varphi\}$ of Max-Keys such that the keys in Σ are applicable to φ and $\Sigma \not\models \varphi$. The idea is to test the algorithm also against sets of Max-Keys $\Sigma \cup \{\varphi\}$ for which it is not only the case that φ is not implied by Σ, but also it is non trivial for the algorithm to determine this fact. The following steps allow us to obtain sets of Max-Keys with these characteristics: a) define a Max-Key φ, b) build its corresponding witness network $G_{\emptyset,\varphi}$, c) add several witness edges to $G_{\emptyset,\varphi}$ taking care of keeping q'_φ not reachable from q_φ, and d) define Max-Keys corresponding to those witness edges. As an example, let us take $\varphi = (\varepsilon, (commune.person.polling, \{district\}))$. The corresponding witness network $G_{\emptyset,\varphi}$ is shown in the first picture of Figure 7. From the witness edges A1, A2 and A3 shown in the second picture of Figure 7, we can derive the following set Σ of Max-Keys among others:

$$\sigma_1 = (commune, (person, \{polling.district\}))$$
$$\sigma_2 = (\_, (\_, \{\_^*.district\}))$$
$$\sigma_3 = (\_^*.person, (polling, \{district\}))$$
$$\sigma_4 = (\_^*.person, (\_, \{\_^*.district\}))$$
$$\sigma_5 = (\_, (\_^*.polling, \{\_\}))$$
$$\sigma_6 = (\_, (\_.\_, \{district\}))$$

where the keys σ_1 and σ_2 correspond to the witness edge A1, the keys σ_3 and σ_4 correspond to the witness edge A2, and the keys σ_5 and σ_6 correspond to the witness edge A3.

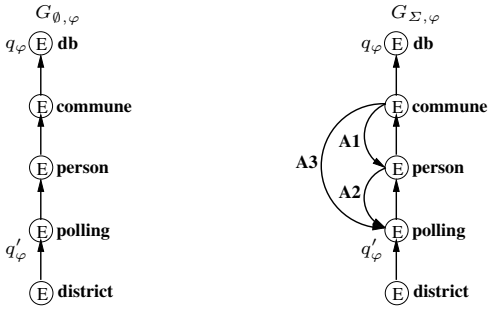


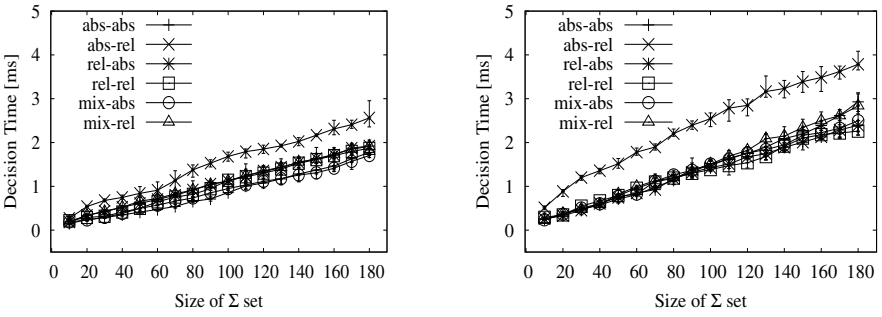
Fig. 7. Witness networks for a non-implied key

The processes described in this section were used to produce a robust collection of Max-Keys as well as a collection of $\mathcal{K}_1$ keys to thoroughly test the performance of the decision algorithm for the implication problem of Max-Key and for the implication problem of $\mathcal{K}_1$ keys, respectively. The results are reported in the next section.

7.2 Performance of the Decision Algorithm for Max-Keys

In order to have a baseline to determine how much the increase in expressibility of the considered class of Max-Keys affects the performance of the decision algorithm, we also include measures of the performance of the decision algorithm in [12] which is optimized for deciding implication of the strictly less expressive fragment $\mathcal{K}_1$ of XML keys from [15].

The results regarding running times for deciding the implication of $\mathcal{K}_1$ keys and Max-Keys are shown in Figures 8(a) and 8(b). In both figures, the x-axis corresponds to the number of keys in Σ, and the y-axis corresponds to the *average* running time required to decide whether Σ implies a given key φ. More precisely, let $time(\Sigma, \varphi)$ be the running time required to decide $\Sigma \models \varphi$ and let Φ be a set of XML keys such that $\Sigma \cap \Phi = \emptyset$, the reported running time corresponds to $\big(\sum_{\varphi_i \in \Phi} time(\Sigma, \varphi_i)\big)/|\Phi|$. In our experiments the sets Φ were composed of 10 fixed XML keys. We tested the scalability of the algorithms by adding, in each iteration, 10 new XML keys to the corresponding Σ sets. Each of the experiments was executed 5 times. The resulting error bars are included in the graphs. They are consistent with time variations commonly produced by the scheduling of the operating system and the use of the *time*() function to measure the experiments [21].



(a) Implication of $\mathcal{K}_1$ keys (b) Implication of Max-Keys

Fig. 8. Performance of the Decision Algorithms for the Implication of XML Keys

We consider Σ sets composed by (i) only absolute keys ("abs"), (ii) only relative keys ("rel") or (iii) both types of keys ("mix"). Given that an input key φ can be either absolute or relative, we have a total of six test cases. The performance shown by the "abs-rel" curves is slightly degraded due to the fact that, in general, the algorithm needs to traverse more nodes to determine whether q'_φ is reachable from q_φ. This is consistent with the way in which the witness networks are defined.

Finally, we present in Figure 9 the aggregate results. For a small set Σ with about 10 XML keys, the execution takes 0.2ms in average, whereas for a large set of about 180 XML keys, the execution takes around 4ms in the worst case. Thus, we can conclude that both decision algorithms are efficient in practice and both scale well. In particular, the price to pay for the added expressibility provided by the Max-Keys is in the order of just 3ms for a considerable big set of 180 Max-Keys.

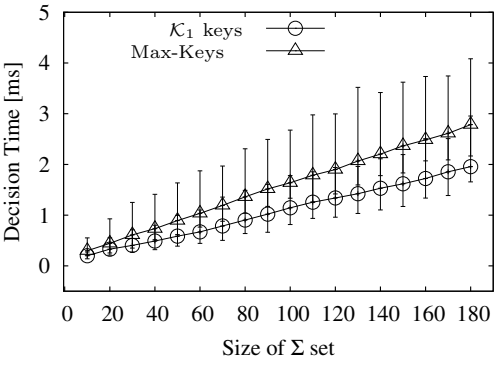


Fig. 9. Aggregate results for the Performance of the Decision Algorithms

7.3 Applying XML Key Reasoning to Document Validation

Fast algorithms for the validation of XML documents against keys are crucial to ensure the consistency and semantic correctness of data stored in databases or exchanged between applications [5]. In this section we propose to use our decision algorithm to compute non-redundant cover sets of Max-Keys. As shown in the experiments this has the potential to significantly reduce the time needed to validate XML documents against sets of Max-Keys. In our experiments we validate XML documents as a whole, starting from scratch. This is clearly useful for instance for tasks such as data cleaning, and it is enough to showcase a concrete example in which our algorithm for the implication of XML keys can be sucessfully used in practice. Nevertheless we note that the central case of incremental validation, which is left out of the scope of this paper, can also benefit from this idea.

Cover Sets for XML Keys. We define the concept of cover set of XML keys following the notion given in [19] for functional dependencies in the relational model. Thus, two sets Σ_1 and Σ_2 of XML keys are a *cover* of one another if they imply exactly the same set of XML keys, i.e., if $\Sigma_1^* = \Sigma_2^*$. A set Σ_c of XML keys is a *non-redundant cover* if none of its proper subsets is a cover for it.

This is the case if there is no key φ in Σ_c such that $\Sigma_c - \{\varphi\} \models \varphi$. Note that a non-redundant cover set Σ_c of a given set Σ of XML keys has potentially fewer keys than Σ and at the same time, every tree T that satisfies all the keys in Σ_c, also satisfies all the keys in the original set Σ.

Thus, given a set Σ of Max-Keys, we can use our decision algorithm to compute a non-redundant cover set Σ_c for it and then, provided it has fewer keys than the original set Σ, validate the target XML document against Σ_c instead of Σ. As shown in our experiments, this can potentially result in enormous time savings.

It is important to note that a set Σ can contain more than one non-redundant cover set and there can also exist non-redundant cover sets that are not included in Σ. In this work we only consider cover sets that are included in Σ.

Validation against Non-redundant Covers. The aim is to determine the viability of computing non-redundant cover sets to reduce the overall time required to validate XML documents against sets of Max-Keys.

Validating an XML document against a set of XML keys involves checking, for every XML key in the set, whether the document satisfies such key. For this task, we use a semi-naïve algorithm that parses the XML document into a DOM tree and then evaluates the XML keys on the resulting tree, by using XPath queries to express their context, target and key paths. We do not use sophisticated validation algorithms such as [9,18] because they are not suitable in its current form to validate Max-Keys. An adaptation of such kind of algorithms for Max-Keys is a complex task which is out of the scope of this work and non essential to prove our point, that is to prove that reasoning about XML keys brings important benefits in this context. Furthermore, the proposed optimization based on non-redundant cover sets is independent of the particular algorithm used for XML key validation.

Tests Results. Given a set Σ of Max-Keys, we compute a non-redundant cover set Σ_c by simply checking for every key $\varphi_i \in \Sigma$, whether $\Sigma - \{\varphi_i\} \models \varphi_i$ (this step is done by our decision algorithm for the implication problem of Max-Keys). If we find a key φ_i that is implied by $\Sigma - \{\varphi_i\}$, we eliminate it from Σ and continue checking the remaining keys against the resulting set $\Sigma - \{\varphi_i\}$. For comparison purpose, we also compute the set $\mathcal{C}$ of all non redundant covers that are strictly contained in Σ. To obtain $\mathcal{C}$, we first check every singleton subset of Σ, then we check every subset of Σ of cardinality two, and so on till we have checked every strict subset of Σ. Note that once we find a non-redundant cover set $\Sigma_c \subset \Sigma$, we can automatically discard every $S_i \subset \Sigma$ such that $\Sigma_c \subseteq S_i$.

The results obtained from the computation of non-redundant cover sets are summarized in Table 3, were $time(\Sigma_c)$ denotes the time in milliseconds required to compute a non-redundant cover $\Sigma_c \subset \Sigma$, $time(\mathcal{C})$ denotes the time in milliseconds needed to compute the set $\mathcal{C}$ of all non-redundant covers contained in Σ, and $min(\Sigma_c)$ and $max(\Sigma_c)$ denote the cardinality of the smallest non-redundant

Table 3. Computation of Non-redundant Covers

XML document	$\lvert\Sigma\rvert$	$\lvert\Sigma_c\rvert$	$time(\Sigma_c)$	$\lvert\mathcal{C}\rvert$	$time(\mathcal{C})$	$min(\Sigma_c)$	$max(\Sigma_c)$
padron							
doc1	10	5	4.041	8	2896.630	5	5
321gone & yahoo							
doc2 & doc3	13	8	5.388	1	22792.900	8	8
DBLP							
doc4	12	7	5.879	1	9424.210	7	7
nasa							
doc5	12	7	5.469	2	10417.200	7	7
SigmodRecord							
doc6	12	6	4.783	2	10633.400	6	6
mondial							
doc7	12	6	3.078	12	9460.690	6	7

cover and the cardinality of the biggest non-redundant cover contained in Σ, respectively.

Figure 10 shows the optimization achieved by pre-calculating non-redundant cover sets during the validation process of the XML documents in Table 2. In both plots the x-axis represent the documents and the y-axis represent the running time in milliseconds. We use a separate plot for doc1 due to the difference in scale, and consequently in validation time, with respect to the others XML documents. Also due to differences in scale, we had to truncate the bars representing the validation time for doc4, doc5 and doc7 against the full set of XML keys. The actual time required for those validations is indicated on top of the corresponding bars.

This results clearly indicate that the running time required to compute a non-redundant cover set is just a tiny fraction of the overall running time required to validate a single XML document against a key. It also confirms that the proposed optimization of the validation process, based on the pre-calculation of non-redundant covers, can significantly reduce the time required for this task.

Significant time savings are achieved in the validation of doc1, doc4, doc5 and doc7. This coincides with the largest tested documents. For instance, while it takes 38 minutes to validate the XML document *padron.xml* (doc1) against 10 Max-Keys, the validation against the best cover in $\mathcal{C}$ only requires 20 minutes.

For reference, we list next the set Σ of Max-Keys used for the validation experiments with the document *padron.xml*. Note that the nodes *given*, *father* and *mother*, which are not included in the simplified tree in Figure 6, refer to the child nodes of the *name* node in the actual structure of the XML document padron.xml. The labels *father* and *mother* refer to the father's surname and the mother's surname, respectively. The keys used in the validation experiments with the others XML documents tested in this work, can be found online as

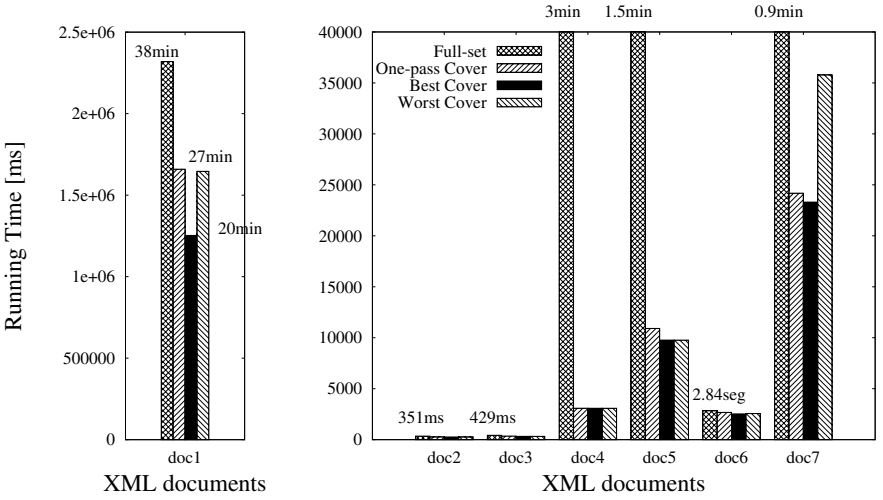


Fig. 10. Validation of XML documents

indicated at the beginning of this section. For the sake of presentation, we omit them here.

1. $(\varepsilon, (commune, \{name.given, name.father, name.mother\}))$
2. $(\varepsilon, (commune, \{\_^*, person\}))$
3. $(\varepsilon, (commune, \{name.given, name.father, name.mother, person\}))$
4. $(\varepsilon, (\_, \{person\}))$
5. $(\varepsilon, (commune, \{person.id\}))$
6. $(\varepsilon, (commune, \{person.id, person.gender\}))$
7. $(\varepsilon, (commune, \{\_id, \_gender, \_name\}))$
8. $(\varepsilon, (commune, \{person.id, person.gender, person.name.given, person.name.father, person.name.mother\}))$
9. $(\varepsilon, (\_^*, \{person.id, person.gender, person.name, person.address\}))$
10. $(\varepsilon, (commune, \{person.id, person.gender, person.name.given, person.name.father, person.name.mother, person.address\}))$

The result of computing all non-redundant covers included in Σ is a collection $\mathcal{C}$ with 8 sets, shown in Table 4. The last set contains simpler keys (i.e., with lower length and fewer single-label and variable-length wildcards), which results in less complex XPath queries and the minimum time required for validation. Specifically, the more complex Max-Keys like $(\varepsilon, (comuna, \{\_^*, person\}))$ involve 345 *commune* nodes, that in total compress $13,248,351$ *person* (complex) nodes with various element and attribute children. Moreover, if we discard the absolute Max-Key with a variable-length wildcard in its target path (which involves $3,239,791$ nodes), we can reduce the validation time to six minutes. Indeed, when more nodes are selected by the target path and more complex nodes by the key paths, then it becomes more time consuming to compute the corresponding value intersection among all pairs of elements.

Table 4. Non-redundant covers used in the validation experiments with *padron.xml*

$(\varepsilon, (commune, \{name.given, name.father, name.mother, person\}))$ $(\varepsilon, (\_, \{person\}))$ $(\varepsilon, (commune, \{\_.id, \_.gender, \_.name\}))$ $(\varepsilon, (\_^*, \{person.id, person.gender, person.name, person.address\}))$ $(\varepsilon, (commune, \{person.id, person.gender, person.name.given,$ $person.name.father, person.name.mother, person.address\}))$
$(\varepsilon, (commune, \{name.given, name.father, name.mother, person\}))$ $(\varepsilon, (\_, \{person\}))$ $(\varepsilon, (commune, \{\_.id, \_.gender, \_.name\}))$ $(\varepsilon, (\_^*, \{person.id, person.gender, person.name, person.address\}))$ $(\varepsilon, (commune, \{person.id, person.gender, person.name.given,$ $person.name.father, person.name.mother\}))$
$(\varepsilon, (commune, \{name.given, name.father, name.mother, person\}))$ $(\varepsilon, (\_, \{person\}))$ $(\varepsilon, (commune, \{person.id, person.gender\}))$ $(\varepsilon, (commune, \{\_.id, \_.gender, \_.name\}))$ $(\varepsilon, (\_^*, \{person.id, person.gender, person.name, person.address\}))$
$(\varepsilon, (commune, \{name.given, name.father, name.mother, person\}))$ $(\varepsilon, (\_, \{person\}))$ $(\varepsilon, (commune, \{person.id\}))$ $(\varepsilon, (commune, \{\_.id, \_.gender, \_.name\}))$ $(\varepsilon, (\_^*, \{person.id, person.gender, person.name, person.address\}))$
$(\varepsilon, (commune, \{name.given, name.father, name.mother\}))$ $(\varepsilon, (\_, \{person\}))$ $(\varepsilon, (commune, \{\_.id, \_.gender, \_.name\}))$ $(\varepsilon, (\_^*, \{person.id, person.gender, person.name, person.address\}))$ $(\varepsilon, (commune, \{person.id, person.gender, person.name.given,$ $person.name.father, person.name.mother, person.address\}))$
$(\varepsilon, (commune, \{name.given, name.father, name.mother\}))$ $(\varepsilon, (\_, \{person\}))$ $(\varepsilon, (commune, \{\_.id, \_.gender, \_.name\}))$ $(\varepsilon, (commune, \{person.id, person.gender, person.name.given,$ $person.name.father, person.name.mother\}))$ $(\varepsilon, (\_^*, \{person.id, person.gender, person.name, person.address\}))$
$(\varepsilon, (commune, \{name.given, name.father, name.mother\}))$ $(\varepsilon, (\_, \{person\}))$ $(\varepsilon, (commune, \{person.id, person.gender\}))$ $(\varepsilon, (commune, \{\_.id, \_.gender, \_.name\}))$ $(\varepsilon, (\_^*, \{person.id, person.gender, person.name, person.address\}))$
$(\varepsilon, (commune, \{name.given, name.father, name.mother\}))$ $(\varepsilon, (\_, \{person\}))$ $(\varepsilon, (commune, \{person.id\}))$ $(\varepsilon, (commune, \{\_.id, \_.gender, \_.name\}))$ $(\varepsilon, (\_^*, \{person.id, person.gender, person.name, person.address\}))$

8 Conclusion

Our contribution is two-fold. Firstly, we introduced an expressive fragment of XML keys (Max-Keys) that is sufficiently flexible to advance XML data processing in important areas of XML application such as consistency management, data integration, query optimization and view maintenance. The flexibility results from the right balance between expressiveness and efficiency of maintenance. Secondly, we have shown through extensive experimentation that reasoning about this expressive fragment of XML keys can be done efficiently in practice, and scales well. Our results promote the use of XML keys to real-world XML practice, where a little more semantics makes applications a lot more effective.

Additionally, we have shown that our contribution to the problem of deciding implication is not only of interest for the problem itself but has immediate consequences for other perennial tasks in XML database management. As an example we have studied the problem of validating an XML document against a set of XML keys. We have presented an optimization method for this validation that computes a non-redundant cover for the set of XML keys given as input so that satisfaction only needs to be checked for the keys in this cover. This can reduce the number of keys significantly, and our experiments show that enormous time savings can be achieved in practice. This holds true even though the validation procedure is able to decide value equality among element nodes with complex content as this is required for the XML keys studied here (and distinguishes them from the keys defined in XML Schema). We would like to emphasize that the use of non-redundant covers does not depend on the particular choice of the XML fragment but can be tailored to any class of XML constraints for which the implication problem can be solved efficiently.

The experiments with large sets of non-trivial XML keys and large XML documents provide a good platform (and also pinpoint the need) for further research in the area. This can go into various directions. XML practice might well warrant the study of other classes of XML keys that require different paradigms to select and compare nodes, or specify restrictions. It would be interesting to investigate the interaction of XML keys with schema specification languages and other classes of database constraints, including functional, multivalued and inclusion dependencies. This is likely to be a challenging task as already observed and illustrated by examples in previous work [8]: keys can non-trivially interact with content models and thus behave differently under such specifications.

It would also be interesting to explore other practical applications of the decision algorithm for the implication problem in areas such as optimization of XPath queries, XML constraint mining, and XML design. The broad area in which XML keys can be applied, as indicated in several parts of this article, warrant further studies.

Acknowledgement. This paper was funded in part by Science Foundation Ireland under Grant No. SFI/08/CE/I1380 (Lion-2). The authors would like to thank the anonymous reviewers for their comments and suggestions, which have contributed to improve the paper. Finally, the authors wish to acknowledge the

contribution of the NeSI high-performance computing facilities and the staff at the Centre for eResearch at the University of Auckland. New Zealand's national facilities are provided by the New Zealand eScience Infrastructure (NeSI) and funded jointly by NeSI's collaborator institutions and through the Ministry of Business, Innovation and Employment's Infrastructure programme (http://www.nesi.org.nz).

References

1. Abiteboul, S., Hull, R., Vianu, V.: Foundations of Databases. Addison-Wesley (1995)
2. Aho, A., Ullman, J., Hopcroft, J.: Data structures and algorithms. Addison-Wesley (1983)
3. Apparao, V., et al.: Document object model (DOM) level 1 specification, W3C recommendation (1998), http://www.w3.org/TR/REC-DOM-Level-1/
4. Arenas, M., Fan, W., Libkin, L.: What's hard about XML schema constraints? In: Hameurlain, A., Cicchetti, R., Traunmüller, R. (eds.) DEXA 2002. LNCS, vol. 2453, pp. 269–278. Springer, Heidelberg (2002)
5. Arenas, M., Libkin, L.: XML data exchange: Consistency and query answering. J. ACM 55, 7:1–7:72 (2008)
6. Bray, T., Paoli, J., Sperberg-McQueen, C.M., Maler, E., Yergeau, F.: Extensible markup language (XML) 1.0, 4th edn., W3C recommendation (2006), http://www.w3.org/TR/xml
7. Buneman, P., Davidson, S., Fan, W., Hara, C., Tan, W.: Keys for XML. Computer Networks 39(5), 473–487 (2002)
8. Buneman, P., Davidson, S., Fan, W., Hara, C., Tan, W.: Reasoning about keys for XML. Inf. Syst. 28(8), 1037–1063 (2003)
9. Chen, Y., Davidson, S., Zheng, Y.: Xkvalidator: a constraint validator for XML. In: CIKM 2002: Proceedings of the 2002 ACM CIKM International Conference on Information and Knowledge Management, pp. 446–452. ACM (2002)
10. Clark, J., DeRose, S.: XML path language (XPath) version 1.0, W3C recommendation (1999), http://www.w3.org/TR/xpath
11. Ferrarotti, F., Hartmann, S., Link, S., Wang, J.: Promoting the semantic capability of XML keys. In: Lee, M.L., Yu, J.X., Bellahsène, Z., Unland, R. (eds.) XSym 2010. LNCS, vol. 6309, pp. 144–153. Springer, Heidelberg (2010)
12. Ferrarotti, F., Hartmann, S., Link, S., Marin, M., Muñoz, E.: Performance analysis of algorithms to reason about XML keys. In: Liddle, S.W., Schewe, K.-D., Tjoa, A.M., Zhou, X. (eds.) DEXA 2012, Part I. LNCS, vol. 7446, pp. 101–115. Springer, Heidelberg (2012)
13. Gottlob, G., Koch, C., Pichler, R.: Efficient algorithms for processing XPath queries. Trans. Database Syst. 30(2), 444–491 (2005)
14. Hartmann, S., Köhler, H., Link, S., Trinh, T., Wang, J.: On the notion of an XML key. In: Schewe, K.-D., Thalheim, B. (eds.) SDKB 2008. LNCS, vol. 4925, pp. 103–112. Springer, Heidelberg (2008)
15. Hartmann, S., Link, S.: Efficient reasoning about a robust XML key fragment. ACM Trans. Database Syst. 34(2) (2009)
16. Hartmann, S., Link, S.: Expressive, yet tractable XML keys. In: EDBT 2009: 12th International Conference on Extending Database Technology. ACM International Conference Proceeding Series, vol. 360, pp. 357–367. ACM (2009)

17. Jungnickel, D.: Graphs, Networks and Algorithms. Springer (1999)
18. Liu, Y., Yang, D., Tang, S., Wang, T., Gao, J.: Validating key constraints over XML document using XPath and structure checking. Future Generation Comp. Syst. 21(4), 583–595 (2005)
19. Maier, D.: Minimum Covers in the Relational Database Model. J. ACM 27, 664–674 (1980)
20. Miklau, G., Suciu, D.: Containment and equivalence for a fragment of XPath. J. ACM 51(1), 2–45 (2004)
21. Stewart, D.B., Khosla, P.K.: Mechanisms for Detecting and Handling Timing Errors. Commun. ACM 40(1), 87–93 (1997)
22. Suciu, D.: XML Data Repository, University of Washington (2002), http://www.cs.washington.edu/research/xmldatasets/www/repository.html
23. Thompson, H., Beech, D., Maloney, M., Mendelsohn, N.: XML Schema Part 1: Structures, 2nd edn., W3C Recommendation (2004), http://www.w3.org/TR/xmlschema-1/

ALACRITY: Analytics-Driven Lossless Data Compression for Rapid In-Situ Indexing, Storing, and Querying

John Jenkins[1,2,⋆], Isha Arkatkar[1,2,⋆], Sriram Lakshminarasimhan[1,2], David A. Boyuka II[1,2], Eric R. Schendel[1,2], Neil Shah[1,2], Stephane Ethier[3], Choong-Seock Chang[3], Jackie Chen[4], Hemanth Kolla[4], Scott Klasky[2], Robert Ross[5], and Nagiza F. Samatova[1,2,⋆⋆]

[1] North Carolina State University, NC 27695, USA
[2] Oak Ridge National Laboratory, TN 37831, USA
[3] Princeton Plasma Physics Laboratory, Princeton, NJ 08543, USA
[4] Sandia National Laboratory, Livermore, CA 94551, USA
[5] Argonne National Laboratory, Argonne, IL 60439, USA
samatova@csc.ncsu.edu

Abstract. High-performance computing architectures face nontrivial data processing challenges, as computational and I/O components further diverge in performance trajectories. For scientific data analysis in particular, methods based on generating heavyweight access acceleration structures, e.g. indexes, are becoming less feasible for ever-increasing dataset sizes. We present ALACRITY, demonstrating the effectiveness of a fused data and index encoding of scientific, floating-point data in generating lightweight data structures amenable to common types of queries used in scientific data analysis. We exploit the representation of floating-point values by extracting significant bytes, using the resulting unique values to bin the remaining data along fixed-precision boundaries. To optimize query processing, we use an inverted index, mapping each generated bin to a list of records contained within, allowing us to optimize query processing with attribute range constraints. Overall, the storage footprint for both index and data is shown to be below numerous configurations of bitmap indexing, while matching or outperforming query performance.

1 Introduction

Increasingly complex simulation models, capable of using high-end computing architectures, are being used to simulate dynamics of various scientific processes with a high degree of precision. However, coupled with this opportunity to augment knowledge and understanding of the highly complex processes being studied are the challenges of conducting exploratory data analysis and knowledge discovery. Specifically, data size on the tera- and peta-scale is becoming a limiting factor in understanding the phenomena latent in these datasets, especially in a post-processing context.

Due to massive dataset sizes, full context analysis is a crucial bottleneck in the knowledge discovery pipeline, being restrained by the limits of computer memory and

⋆ Authors contributed equally.
⋆⋆ Corresponding author.

A. Hameurlain et al. (Eds.): TLDKS X, LNCS 8220, pp. 95–114, 2013.

I/O bandwidth. Most commonly, applications of which such data exploration processes are characteristic are interactive and require near-real-time I/O rates for full data exploration. However, I/O access rates are too slow to support efficient random disk access in real-time for large-scale data sets, necessitating new approaches to reduce the I/O pressure of extreme-scale data analytics.

A *knowledge priors* approach to data analytics is promising in restricting data to smaller and more practical sizes. Often times, scientists have some prior knowledge about the regions of interest in their data. For example, fusion scientists aiming to understand plasma turbulence might formulate analysis questions involving correlations of turbulence intensities in different radial zones ($0.1 < \psi < 0.15; 0.3 < \psi < 0.35; 0.5 < \psi < 0.55; 0.7 < \psi < 0.75; 0.9 < \psi < 0.95$). Likewise, climate scientists aiming to understand factors contributing to natural disasters might limit their search to particular regions or perhaps only a single region.

Thus, formulating queries on scientific simulation data constrained on variables of interest is an important way to select interesting or anomalous features from large-scale scientific datasets. Traditional database query semantics are an effective means to express such queries. This allows us to leverage a great deal of work from the database community on query processing. The indexing techniques used in traditional database systems, such as $B-$trees [9] and bitmap indexes [23], have been used extensively in the literature. However, while indexing is a blessing for fast and efficient query processing, it is arguably a curse in terms of storage; index sizes are often 100-300% of the original column size for high-cardinality data (such as double-precision data) [26], which is a huge bottleneck for storage- and I/O-bound extreme-scale applications.

A number of bitmap index compression techniques have been introduced to reduce the size of the bitmap index while maintaining fast query retrieval. In particular, Word Aligned Hybrid (WAH) [24] bitmap compression is used in FastBit [23], a state-of-the-art scientific indexing technology with fast query processing capabilities. Notably, however, the total storage footprint for a high-cardinality data column along with an associated FastBit index is around 200% of the original size [25], which is still prohibitive in many extreme-scale contexts. Furthermore, while this indexing scheme is optimized for region-retrieval queries over spatio-temporal data sets (i.e., returning the record IDs/regions that match a query constraint), returning the *actual values* of the variables associated with these regions (i.e. value retrieval query) is equally important in data analytics, necessitating an expanded approach.

Therefore, we present ALACRITY, an Analytics-driven Lossless Compression methodology, for Rapid in-situ Indexing, sToring, and querYing. ALACRITY integrates data reduction and indexing methodology for floating-point datasets, optimized for query-driven data analytics over scientific data. We believe that a tight cohesion between the data and index allows us to optimize storage requirements while at the same time facilitating both fast indexing at simulation-time and range query processing with value retrieval during analysis. In particular, our focus is on write-once, read-many (WORM) datasets utilizing double-precision floating-point variables, as are commonly produced by large-scale, high-fidelity simulation runs and subsequently analyzed by numerous application scientists in multiple (often global) contexts. A few examples of such data are the particle-based fusion simulation GTS [20] and the direct numeri-

cal combustion simulation S3D [8], each of which are comprised of primarily double-precision, high-cardinality variables ($\approx$ 100% unique values for GTS, $\approx$ 50% unique values for S3D).

Toward the goal of developing a system given this motivation, we make the following contributions in this paper:

- We present a lossless compression methodology for floating-point (single and double-precision) variables utilizing unique-value encoding of the most significant bytes. Our lossless compression reduces the size of a number of high-entropy, double-precision scientific datasets by at least 15%. Compared to lossless compression techniques like FPC [6], optimized for floating-point data, we report superior average compression ratios.
- Using our lossless compression method, we optimize range query evaluation including value retrieval by binning the column data by the distinct significant byte metadata, integrating efficient compressed-data organization and decompression of retrieved results. Compared to state-of-the-art techniques like FASTBIT [23], we provide comparable or better performance on range queries retrieving record IDs. For range queries that additionally retrieve variable values, we achieve up to one order of magnitude improvement in performance.
- For query processing, we utilize an inverted index, incurring a smaller storage footprint compared to other database indexing schemes. Using an inverted index compression via the PForDelta algorithm [30], we achieve a combined index and data column size of only 77–93% of the original column size.

2 Background

2.1 Indexing

Search and query processing operations on traditional database systems like Oracle, MySQL, and DB2 involve the use of indexing techniques that are usually variants of either bitmap indexes or $B-$trees. While these techniques are effective in speeding up query response times, they come at the cost of a heavy-weight index management scheme. Indexing with $B-$trees [9], which tends to be more suitable for transactional databases that require frequent updates, is observed to consume storage that is one-to-three times the size of the raw column data for high-cardinality attributes. Scientific data, which is typically read (or append) only, have been shown to be better served with bitmap-based indexing techniques [19, 23], providing faster response times with lower index storage overhead.

While there are numerous technologies that use variants of bitmap indexing, we primarily focus on FASTBIT [23], a state-of-the-art bitmap indexing scheme, that is used by a number of scientific applications for answering range queries. FASTBIT employs a Word-Aligned-Hybrid (WAH) compression scheme based on run-length encoding, which decreases the index storage requirement and allows FASTBIT to perform logical operations efficiently on the compressed index and compute partial results by scanning the index. While the required storage for WAH is larger than that for bitmap compression variants such as Byte-aligned Bitmap Compression [4], WAH has been shown to be

far faster for query processing. For those records that cannot be evaluated with the index alone, FASTBIT resorts to performing a read of the raw data, in what is called *candidate checks*. Unfortunately, the bitmap index created is sensitive to the distribution and cardinality of the input data, taking anywhere from 30 to 300% of the raw column size. The space can partly be reduced through techniques such as *precision binning*, at the cost of disturbing the distribution of values along the bins.

Another type of index, popular for document indexing, is an inverted index [22, 29]. Traditionally, in document clustering systems, a single document is identified by terms, which typically correspond to some subset of written language. An inverted index, in this case, maps each term in the dictionary to the list of documents the term appears in, greatly speeding up queries constrained by term. In this work, an inverted mapping is used in a different context: we are mapping histogram bins to lists of records that fall within each bin.

2.2 Compression

Data compression methods within databases have been widely studied as an important component for lowering the storage footprint of data-stores [11, 14, 21]. For example, the column-oriented database C-Store [2] uses null compression (elimination of zeroes), dictionary encoding, and run-length encoding for effective data reduction of attributes organized contiguously, as opposed to the traditional row-store organization. While these methods have limited use on floating-point data due to high-entropy significand bits, our work does share similarity with the dictionary encoding method, in that we compress floating-point data through identifying unique values and assigning them reduced bitwise representations. However, we perform this on only the most significant few bytes of the floating-point data, as opposed to the full dataset as in C-Store, and discard the representation entirely when using the inverted index for our query processing methodology.

A compression methodology particularly important to our methods is based on inverted index compression techniques. Asides from general purpose compressors, specialized techniques, such as Simple9, Simple16, Relate10 and Carryover12 [3] provide high compression ratios while being computationally efficient for compressing and processing indexes on text collections. Furthermore, document reordering methods have been devised to increase locality, thereby achieving higher compression ratios [27]. For our method, a particularly important inverted index compression algorithm is the PFor family of compressors, which include PFor, PForDelta, and PDict [30], built specifically for fast compression and decompression speeds on modern CPU architectures. Each method uses reduced, fixed bit widths to encode values with a high degree of similarity (PFor and PForDelta) or commonly occuring values (PDict), encoding the remaining in full-precision as *exceptions*. Of these methods, PForDelta is the basis for our inverted index compression, as it first encodes differences between successive values before compressing. See Section 3.3.

As mentioned, many general-purpose and specialized compression methodologies fail to provide high compression ratios on floating-point data. Part of the reason for this is that floating-point scientific data is notoriously difficult to compress due to high entropy significands, of which floating-point data is primarily composed of (23 of 32

bits for single precision and 52 of 64 bits for double-precision). Much work has been done to build compressors for these kinds of data, mostly based on difference coding. Algorithms such as FPC [6] and fpzip [17] use *predictors* like the Lorenzo predictor [12], FCM [28] and DFCM [10] to compress. Given an input stream of floating-point values, the predictors use the previously seen values to predict the next value in the stream, and rather than attempt to compress the floating-point values themselves, the compression algorithm uses a measure of error between the predicted and actual value, typically as an XOR operation.

Our compression/indexing methodology is based on treating the most significant bytes of floating-point data differently than the least significant bytes. Isenburg *et al.* use the same underlying concept in a prediction-based compression utility, which partitions the sign, exponent, and significand bits of the prediction error, followed by compression of each component [13]. Unlike their method, our method must maintain the approximability of floating point datasets by treating the most significant bytes as a single component (sign, exponent, and the most significant significand bits), enabling efficient index generation and range query processing over the compressed data.

Another method that is based on processing data with respect to significant bytes is ISOBAR preconditioner [18]. Based on the observation that a significant byte-wise view of the data can yield patterns not picked up by existing compressors, ISOBAR first determines the *compressibility* of input data by looking at the frequency distribution on a significant byte level. Significant byte columns that appear to have a uniform frequency distribution (such as mantissa bytes in floating-point variables) are ignored in subsequent compression, leading to greatly increased compression speeds.

3 Method

3.1 System Overview

As mentioned, the goal of this paper is to facilitate query-driven analysis of large-scale scientific simulation data with storage-bound requirements. There are two stages where we focus our design to achieve this goal: first, while simulation data is being generated and is still in memory, or later as a post-processing step, we can process and reorganize a floating-point dataset to compress the data. Second, we can modify the new organization of data to optimize query processing on the preprocessed data. For this purpose, we introduce two components in the scientific knowledge discovery pipeline, the *lossless compressor/indexer* and the *query engine*. These correspond to two different use cases using the same underlying process – a compression-only use case and a query-processing use case.

3.2 Compression

Scientific simulations use predominantly double-precision floating-point variables, so the remainder of the paper will focus on compression and query processing for these variables, though our method can be applied to floating point numbers of different precision. The underlying representation of these variables, the IEEE 754 floating-point

standard [1], is a primary driver of our compression and querying methodology, so we briefly review it here. The standard encodes floating point values using three components: a *sign* bit, an *exponent* field, and a *significand* field. For example, 64-bit double-precision values use one sign bit, 11 exponent bits, and 52 significand bits. Given the sign bit s, the unsigned integral representation of the exponent field e, and each significand bit m_i (most to least significant), the resulting value encoded by a double-precision variable is:

$$\text{value} = (-1)^s \times 2^{e-1023} \times (1 + \sum_{i=1}^{52} (m_i 2^{-i})). \quad (1)$$

Note that, all other components being equal, a difference of one in the exponent fields of two double-precision numbers leads to a 2x difference in the represented values.

Our key observation for the compression process is that there is similarity among values in our target datasets with respect to orders of magnitude. For instance, in a simulation grid, adjacent grid values are unlikely to differ in orders of magnitude, except perhaps along simulation-specific phenomenon boundaries. Furthermore, the encoding naturally lends itself to accurate approximation given the exponent components. Hence, we base our compression and query processing methodology on the commonality in the sign and exponent field of double-precision datasets.

Figure 1 gives an overview of the compression process, developed under the assumption of similar exponent components and with the goal of being amenable to range query processing. For an N-element *partition* (a single block of configurable size from the dataset, to be compressed and indexed as a unit), we split the $8N$-byte double-precision column stream into two components: a kN-byte *high-order byte stream* consisting of the most significant k bytes of each value, and the remaining $(8-k)N$-byte *low-order byte stream* consisting of the remaining significant bytes. Using the observation of highly similar sign and exponent values, we identify the unique high-order bytes and discard redundant values. Let n be the number of unique high-order byte patterns. We define a *bin* to be a set of low-order bytes with equivalent high-order bytes, with bin edges

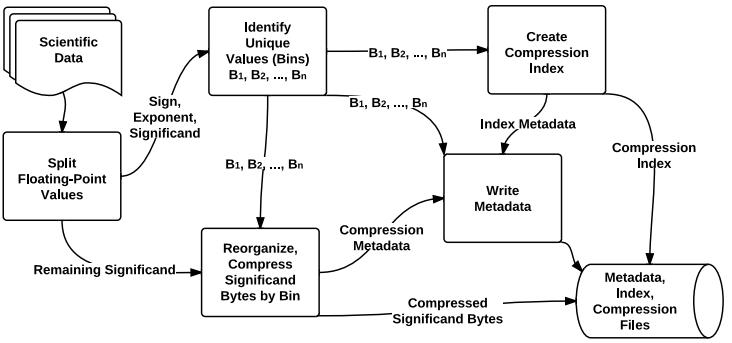


Fig. 1. Various stages of the compression methodology, described in Section 3.2. The bitmap index is used for compression, while the inverted index is used in query processing.

$B_1, B_2, \ldots, B_n$ corresponding to the sorted unique patterns. Figure 2 shows the relationship between floating-point values, the high and low-order bytes, and their resulting bins. The low-order bytes are reorganized into their respective bins, and a record ID (RID) to bin mapping M is generated to maintain the original organization, using a bitmap with $\lceil \log(n) \rceil$ bits per identifier. A general-purpose compressor (such as bzip2) is then run on the bin mapping M, as well as (optionally) the low-order bytes.

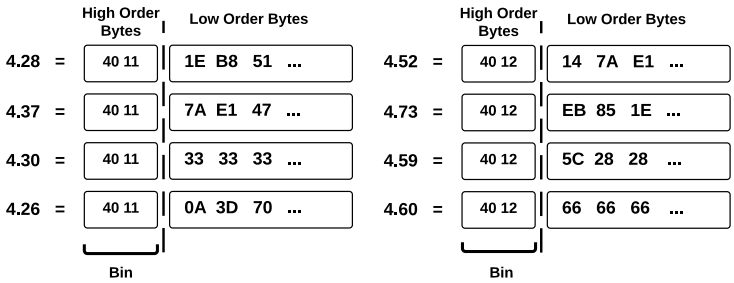


Fig. 2. Mapping between floating-point numbers, high- and low-order bytes, and their respective bins

Three data structures are produced as the result of the compression process: (1) the compression metadata, defining the high-order byte values and file offsets of each bin, (2) the compressed RID-to-bin mapping M, and (3) the bin-organized low-order bytes.

The value of k should be chosen with two goals in mind: to cause the number of distinct high-order bytes to stabilize with an increasing stream size, and to maximize the redundancy of the patterns (for compression) while encoding the entirety of the sign and exponent components (for future query processing). For scientific floating point data, we have found $k = 2$ to be the most effective; it covers the sign bit, all exponent bits, and the first four significand bits of double-precision values (approximately two significant figures in base 10 scientific notation). This makes sense, as higher degrees of precision in scientific data tend toward high-entropy values. To verify our choice of k for this paper, Figure 3 shows the number of distinct high-order bytes recorded as a

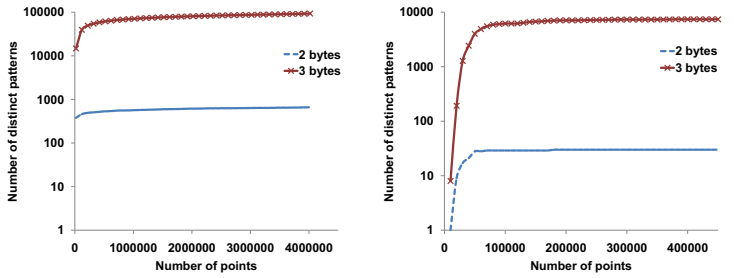


Fig. 3. Cumulative growth of the number of distinct higher order 2-byte and 3-byte pattern for increasing data size

data stream is processed. For both $k = 2$ and 3, a relatively small cardinality is seen relative to the number of points processed, with the distinct values quickly reaching a (near) maximum.

Recall that the metadata consists of unique high-order bytes as well as their respective file offsets to the low-order byte payload. Hence, the metadata size is directly proportional to the number of unique high-order bytes. As shown in Figure 3, for two of the scientific datasets, the size of metadata is less than 0.1% of the dataset for $k = 2$, due to the small number of distinct patterns. For $k = 3$, however, the number of distinct patterns increases by a factor of 100 due to the addition of the higher-entropy significand bits. This increases the metadata size similarly, while additionally increasing the size of the RID to bin mapping logarithmically. Given the trends in Figure 3, we expect random sampling to be sufficient to determine a good value of k for double-precision datasets.

3.3 Query Processing: Index Generation

The compression methodology presented in Section 3.2 is, as will be shown, effective at improving the compression ratio of many scientific datasets, but is not optimized for query processing. If a range query is performed using our compression index, the entire RID-to-bin mapping M would need to be traversed to map the binned data back to RIDs. Thus, we develop another method to optimize for range queries by using an inverted index, at the cost of additional storage. This inverted index, which we denote as M^{-1}, maps each bin to a list of RIDs sharing the same high-order bytes, creating a *bin-based value-to-RID* mapping. Figure 4 illustrates the index used in compression compared to the inverted index. This organization is advantageous for range query processing because we now access the RIDs by bin (the same access pattern as with the low-order bytes). It is initially disadvantageous, however, because of the increased space. This means, for a partition of N elements, approximately $N\log(N)$ bits are needed to store the index, with marginal additional space to store metadata such as the number of elements within each bin. Bounding the maximum partition size to 32GB of double-precision data ensures that each RID in the inverted index needs no more than four bytes, making the index size less than 50% of the raw column size, or lower for smaller partitions. As a simple example, a partition size of 2GB of double-precision data requires 28 bits for each RID, translating to an index size of 43.75% of the raw column size. This is assuming, of course, that the partition is completely filled.

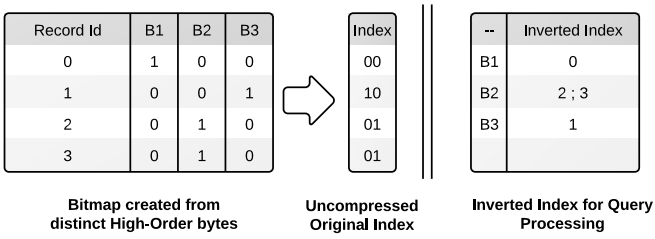


Fig. 4. Building an inverted index for query processing from the index used in compression

Inverted Index Compression. While the index is relatively small compared to the column size, we make a few observations that allow us to further reduce index storage overhead. Our inverted index works on the bin-level by using linearized RIDs, with the resulting structure of an ordered list of RIDs. This presents the perfect opportunity to use a compressed inverted index, based on difference (delta) encoding. Specifically, we use the PForDelta algorithm. Given a desired bits-per-item parameter b, PForDelta stores a *base* value (the first RID in the sorted list), and differences between consecutive elements, using only b bits per-difference. For values that cannot be stored using b bits (namely, differences greater than $2^b - 1$), that element, called an *exception*, is stored at the end of the compression block, with a special marker put in it's place. Then, for decompression, a two-pass approach is taken: the first pass restores all deltas to their original state, and the second corrects, or "patches" the values encoded as exceptions, while computing the running sum and restoring values. The PForDelta default block size is used for compression (128 elements), for both cache efficiency and to allow the value of b to vary across blocks.

For PForDelta compression to be effective, the parameter b must be determined to optimize and balance compression ratio and speed. While we do not provide a detailed evaluation, we use the approach from the original PForDelta paper: select the b that results in the highest compression ratio by computing, for each compression block, the value of b that minimizes total size (compressed delta list + exception list). For the datasets evaluated in this paper, this heuristic represents a good tradeoff between compression ratio and speed, because a large number of exceptions hurts both metrics.

Low-order Byte Compression with ISOBAR. The need for fast (de)compression speeds in the indexing process requires us to revisit the idea of low-order byte compression – for the compression target of our method, the primary metric is instead storage reduction. Previous work [15] showed two trends using low-order byte compression along with indexing: that compression gains for the low-order bytes vary widely across datasets, and that compression speed was the limiting factor for indexing speed. We believe these trends arise because the underlying data being compressed is composed of entirely floating-point mantissa bytes, which tend to have a more uniform distribution.

Based on these observations, we use the ISOBAR preconditioner to cut down on compression costs, while still receiving the benefits of data reduction. Figure 5 shows the modified compression process. Each significant byte-column is analyzed for compressibility using frequency analysis. Each column deemed compressible by ISOBAR is then compressed, while the "incompressible" columns are kept as-is, saving on computation.

3.4 Query Processing: File Layout

The data used by the query processing engine is split into three components: a metadata file, an index file, and a compression file, each corresponding to its purpose described in the previous sections.

The metadata file layout is shown in Figure 6. The metadata file contains partition information, including file offsets for each partition and bin, the number and bounds (high-order bytes) of bins, and the number of values per bin per partition. The index

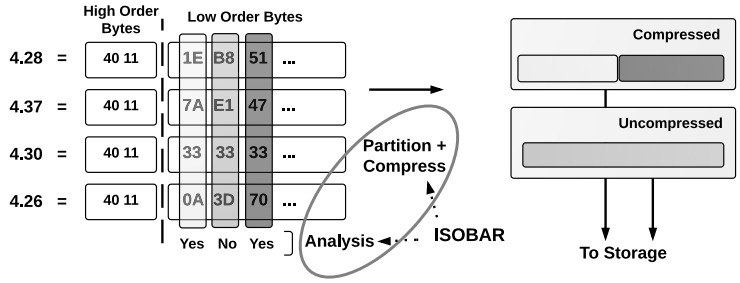


Fig. 5. ISOBAR analysis and compression, applied on a per-bin basis

```
<N number of partitions>
<Metadata offset for partition t> (0 ≤ t < N)
<Index offset, state flag for partition t> (0 ≤ t < N)
<Low order byte offset, state flag for partition t> (0 ≤ t < N)
(Repeat for 0 ≤ t < N)
<P number of elements in partition t>
<B number of bins>
<Number of elements in bin b> (0 ≤ b < B)
<Bin bound b> (0 ≤ b < B)
<Compression offset b> (0 ≤ b < B)
(End Repeat)
```

Fig. 6. Metadata file format

file and the compression file contain the RIDs and compressed low-order bytes, respectively. A single scan of the metadata file is necessary for query processing and is small enough to be held in memory to optimize future queries. In our experimentation, however, we do not consider this possibility.

3.5 Query Processing: Range Queries

The processing of range queries is based on two characteristics of our compression/indexing process: that the bins (low-order bytes and inverted index) are organized on disk in increasing order of high-order bytes, and that bin edges (the high-order bytes) provide a lower bound on the values of RIDs within each bin by treating the high-order bytes as a truncated double-precision value.

The query evaluation process is shown in Figure 7. Given a variable constraint $[v_1, v_2)$, the metadata file shown in Figure 6 is traversed to obtain the necessary high-order bytes and bin file-offsets. Using the high-order bytes as a lower-bound for values within a bin, the boundary bins B_x and B_y are obtained using a binary search. Then, a single, contiguous read is performed per partition in each of the index and low-order bytes files in order to fetch the data corresponding to the range of bins $B_x, B_{x+1}, \ldots, B_y$, taking advantage of the bin organization in file. The column data corresponding to the

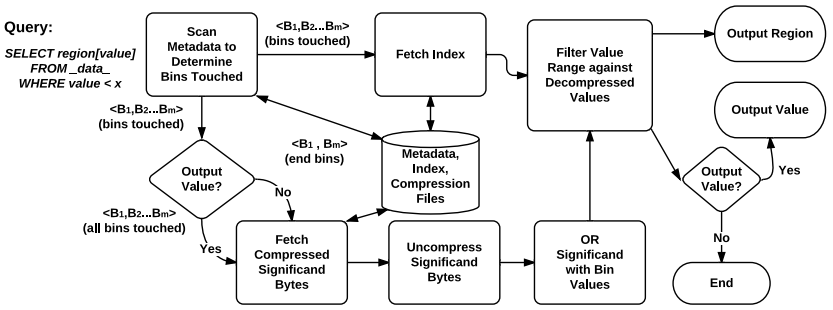


Fig. 7. Query processing methodology, taking into account metadata, index, and compression data fetching and aggregating

low-order bytes are reconstructed and only the data in boundary bins are filtered against the query bounds.

In the case of queries requesting only RIDs, not all of the low-order bytes need to be fetched and reconstructed. Only the bins at each boundary need be read and checked against the query constraints, as all remaining bins are guaranteed to fit within the query bounds.

4 Results and Discussions

4.1 Experimental Setup

We performed our experiments on the Lens cluster at Oak Ridge National Laboratory, dedicated to high-end visualization and data analysis. Each node in the cluster is made up of four quad-core 2.3 GHz AMD Opteron processors and is equipped with 64GB of memory. All experiments were run with data located on the Lustre filesystem. For the indexing and query processing experiments, we compare against WAH encoding as implemented by FASTBIT, version 1.3.4. To avoid database-related overheads such as concurrency control, transaction support, etc. and provide a fair comparison between technologies, we wrote a minimal query driver for FASTBIT using only the necessary indexing and querying functions provided in the FASTBIT API. Also in the interest of fairness, we use the same partition size of 2GB for both our method and FASTBIT.

4.2 Datasets

To evaluate our compression, indexing, and query processing performance, we use a collection of double precision datasets from various sources. The majority of the datasets (*msg*, *num*, and *obs*) are publicly available and discussed by Burtscher and Ratanaworabhan [7]. We additionally use timeslice data for numerous variables generated by the GTS [20], FLASH [5], S3D [8], and XGC-1 [16] simulations.

In particular, we used the following two scientific simulation datasets to evaluate our query performance in terms of value-centric queries and region-centric queries: 1) GTS [20], a particle-based simulation for studying plasma microturbulence in the

core of magnetically confined fusion plasmas of toroidal devices, and 2) S3D [8], a first-principles-based direct numerical simulation (DNS) of reacting flows that aids the modeling and design of combustion devices.

4.3 Query Processing

Index Generation

We evaluate the performance of our index generation methodology with respect to both computational and storage efficiency. We utilize inverted index compression, but not low-order byte compression, when comparing our method to FASTBIT's, as this represents the best tradeoff between indexing speed and storage footprint. Table 1 shows the results obtained from these experiments.

As shown in Table 1, our combined index and data encoding outperforms numerous FASTBIT configurations with respect to both speed and storage. Even with index compression, ALACRITY is shown to encode the data at an order of magnitude higher rate

Table 1. Query index generation throughput and storage footprint. A_I: ALACRITY with inverted index compression. F_D: FASTBIT with default configuration (10^5 bins). $F_{2,3}$: FASTBIT with bin boundaries at two/three significant digits.

Dataset	**Index Gen. (MB/s)**				**Storage (data+index)**			
	In-situ		**Post-proc.**		**Requirement (%)**			
	A_I	F_2	A_I	F_2	A_I	F_2	F_3	F_D
msg_bt	**120**	9	**67**	8	**87.3**	152.0	178.1	192.6
msg_lu	**125**	10	**65**	9	**87.2**	162.6	197.9	201.6
msg_sp	**136**	10	**73**	9	**83.0**	126.2	157.0	197.7
msg_sppm	**137**	12	**73**	11	**81.3**	114.7	116.8	125.3
msg_sweep3d	**140**	8	**63**	7	**85.0**	148.4	187.5	200.9
num_brain	**138**	9	**65**	8	**87.6**	164.3	191.5	202.3
num_comet	**107**	7	**53**	7	**92.9**	181.4	193.1	196.1
num_control	**109**	6	**58**	6	**93.1**	154.8	199.6	200.9
num_plasma	**130**	6	**38**	6	**86.6**	157.3	189.3	197.6
obs_error	**138**	11	**51**	10	**88.5**	149.9	167.6	176.9
obs_info	**132**	11	**28**	10	**85.0**	138.1	181.3	219.3
obs_spitzer	**137**	12	**69**	11	**87.0**	146.4	177.2	198.3
obs_temp	**121**	10	**40**	9	**91.9**	187.0	200.1	210.0
gts_phi_l	**111**	7	**42**	6	**92.9**	181.5	199.4	208.8
gts_phi_nl	**112**	7	**42**	6	**92.9**	183.6	199.7	208.9
gts_chkp_zeon	**110**	7	**28**	7	**91.3**	176.3	198.9	220.4
gts_chkp_zion	**114**	7	**28**	6	**89.9**	166.1	194.6	220.0
gts_potential	**112**	6	**71**	6	**92.5**	184.0	197.9	199.8
xgc_iphase	**105**	10	**68**	9	**90.0**	168.3	172.3	176.9
s3d_temp	**144**	14	**71**	13	**80.4**	117.2	135.4	202.0
s3d_vvel	**123**	11	**64**	10	**90.1**	171.7	195.0	202.1
flash_velx	**101**	9	**80**	8	**82.4**	123.8	157.2	195.7
flash_vely	**107**	9	**83**	9	**79.4**	112.3	137.3	193.1
flash_gamc	**110**	16	**83**	14	**77.5**	100.4	102.1	198.1

than the fastest configuration of FASTBIT we tested, though the gap narrows somewhat when including read/write measurements: computing the compressed bitmap indexes appears to be FASTBIT's rate-limiting factor.

To show a full picture of ALACRITY's performance characteristics with respect to data and index encoding, Table 2 shows indexing results over the same datasets, this time showing the effect of different configurations of ALACRITY. First, the use of index compression is a good way of reducing the overall storage footprint while not incurring high computational costs. Second, the efficacy of low-order byte compression shows widely varying storage results based on the particular underlying dataset. This is a result of the mantissa bytes being highly entropic, though ISOBAR improves the encoding speed when not much data is selected for compression. Third, the choice of k is highly important in the context of indexing performance. As increasing values of k tend to increase the number of unique patterns exponentially (see Figure 3), the resulting

Table 2. Query index generation throughput and storage footprint among multiple ALACRITY configurations. A_{IB}: ALACRITY with inverted index and low-order byte compression. A_B: ALACRITY with low-order byte compression only. A_I: ALACRITY with inverted index compression only. A_{I3}: ALACRITY with inverted index compression and $k = 3$. A: ALACRITY without additional compression.

Dataset	**Index Gen. (MB/s)**										**Storage (data+index) Requirement (%)**				
	In Situ					**Post-proc.**									
	A	A_I	A_{I3}	A_B	A_{IB}	A	A_I	A_{I3}	A_B	A_{IB}	A	A_I	A_{I3}	A_B	A_{IB}
msg_bt	**207**	120	44	28	25	**85**	67	33	23	21	125.0	87.3	97.7	118.7	**81.0**
msg_lu	**217**	125	42	26	24	**82**	65	30	22	20	125.0	87.2	99.6	124.4	**86.6**
msg_sp	**240**	136	57	33	30	**86**	73	42	27	25	125.0	83.0	91.5	120.7	**78.7**
msg_sppm	**224**	137	82	66	54	**86**	73	54	48	43	125.0	81.3	75.1	60.6	**16.8**
msg_sweep3d	**233**	140	56	32	29	**76**	63	38	25	23	125.0	85.0	92.5	105.1	**65.1**
num_brain	**243**	138	58	22	21	**77**	65	39	19	18	125.0	87.6	92.0	124.6	**87.2**
num_comet	**167**	107	26	32	29	**68**	53	20	24	23	125.0	92.9	114.2	114.0	**81.8**
num_control	**179**	109	33	27	24	**71**	58	25	22	20	125.0	93.1	117.2	124.1	**92.2**
num_plasma	**213**	130	66	107	79	**45**	38	30	38	33	125.0	86.6	86.9	50.9	**12.5**
obs_error	**243**	138	62	42	37	**56**	51	37	28	25	125.0	88.5	91.3	88.1	**51.6**
obs_info	**226**	132	37	52	45	**38**	28	17	21	18	125.0	85.0	114.5	77.8	**37.8**
obs_spitzer	**251**	137	59	30	27	**85**	69	41	24	23	125.0	87.0	87.3	94.8	**56.8**
obs_temp	**206**	121	35	25	23	**55**	40	22	17	16	125.0	**91.9**	109.3	125.1	91.9
gts_phi_l	**184**	111	33	36	32	**81**	42	19	23	21	125.0	**92.9**	122.1	125.1	93.0
gts_phi_nl	**186**	112	34	41	36	**88**	42	22	25	24	125.0	**92.9**	117.3	125.0	92.9
gts_chkp_zeon	**181**	110	19	28	26	**83**	27	11	16	15	125.0	**91.3**	136.3	125.2	91.4
gts_chkp_zion	**191**	114	20	28	25	**85**	28	12	17	15	125.0	**89.9**	133.3	125.1	90.0
gts_potential	**181**	112	33	48	40	**87**	71	28	37	33	125.0	92.5	95.5	124.9	**92.4**
xgc_iphase	**150**	105	28	29	26	**80**	68	25	25	23	125.0	90.0	88.0	105.4	**70.4**
s3d_temp	**267**	144	100	48	42	**106**	70	58	35	32	125.0	80.4	76.9	118.7	**74.2**
s3d_vvel	**216**	123	43	23	21	**97**	64	32	19	18	125.0	**90.1**	97.4	125.0	90.2
flash_velx	**267**	142	78	23	21	**101**	80	52	20	19	125.0	82.4	90.1	125.0	**82.4**
flash_vely	**246**	144	80	23	21	**107**	83	54	20	19	125.0	79.4	89.2	125.0	**79.4**
flash_gamc	**270**	148	142	96	74	**110**	83	83	60	54	125.0	77.5	68.7	115.1	**67.6**

effect on indexing performance is negative in both generation time and storage overhead. However, it is useful in query processing, as it minimizes the cost of processing misaligned bins.

In our previous work, we utilized *zlib* for the low-order byte compression. A detailed comparison between the use of ISOBAR and zlib can be seen in Table 3. Note that the underlying compressor used by ISOBAR in this work is actually zlib; ISOBAR is technically a compression preconditioner. The use of ISOBAR on the same datasets produced a speed increase while leaving the storage footprint virtually unchanged. Specifically, we see a median 22% (mean 40%) increase in encoding throughput, with a corresponding median 0.01% (mean −0.31%) increase in storage.

Table 3. ALACRITY indexing performance, using ISOBAR and zlib as the underlying compressors

Dataset	**Index Gen. (MB/s) In Situ**		**Storage (data+index) Requirement (%)**	
	w/ISOBAR	w/zlib	w/ISOBAR	w/zlib
msg_bt	**28**	21	**118.7**	119.4
msg_lu	**26**	21	**124.4**	124.4
msg_sp	**33**	20	**120.7**	124.0
msg_sppm	**66**	37	60.6	**59.6**
msg_sweep3d	**32**	22	105.1	**96.6**
num_brain	**22**	20	124.6	**124.5**
num_comet	**32**	17	**114.0**	116.2
num_control	**27**	21	124.1	**124.1**
num_plasma	**107**	62	**50.9**	51.4
obs_error	**42**	30	**88.1**	94.9
obs_info	**52**	37	77.8	**75.1**
obs_spitzer	**30**	20	94.8	**94.4**
obs_temp	**25**	21	125.1	**125.0**
gts_phi_l	**36**	21	125.1	**125.0**
gts_phi_nl	**41**	21	125.0	**125.0**
gts_chkp_zeon	**28**	21	125.2	**125.1**
gts_chkp_zion	**28**	21	125.1	**125.1**
gts_potential	**48**	20	**124.9**	125.0
xgc_iphase	**29**	22	105.4	**105.3**
s3d_temp	**48**	19	**118.7**	123.3
s3d_vvel	**23**	20	125.0	**125.0**
flash_velx	**80**	21	**125.0**	125.0
flash_vely	**83**	21	**125.0**	125.0
flash_gamc	**83**	17	**115.1**	121.4

End-to-End Query Performance Evaluation

For an end-to-end performance comparison, we perform queries under a number of scenarios, using the GTS potential (gts_potential) and S3D temperature (s3d_temp) variables. We look at two types of range queries: those that return record IDs given constraints on variables (which we refer to as "region-centric" queries, as they are used to

retrieve "regions of interest"), and those that additionally output the values of the variables (which we will refer to as "value-centric" queries). We compare ALACRITY on each of these query types against FASTBIT, which is optimized for range queries, especially those of the "region-centric" type.

For both types of queries, we use ALACRITY with index compression, and ALACRITY without it. For region-centric queries, we use $k = 3$ and FASTBIT precision-3 binning (e.g., bin boundaries use three significant figures). This is so we can avoid performing costly candidate checks in both FASTBIT and ALACRITY and evaluate query processing with only the index, corresponding to fully "aligned" queries. For value-centric queries, we use $k = 2$ and FASTBIT precision-2 binning. This is done because value-centric queries are dominated by data retrieval, and the lower-precision indexes incur lower time to process, at the cost of having higher false positives. As the majority of the data read in tends to satisfy query constraints due to binning, the cost of pruning false positives is outweighed by the benefit of a lighter index.

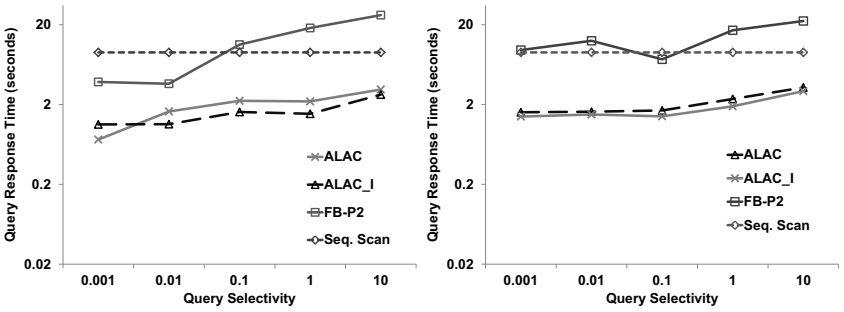


Fig. 8. Comparison of query response times for varying query selectivity, between ALACRITY (ALAC), ALACRITY with compressed inverted indexes (ALAC_I), FASTBIT (FB-P2) sequential scan. The left plot is for S3D temperature, while the right plot is for GTS potential.

Value-centric Queries. Figure 8 shows value-based query response time using our method, compared to FASTBIT's precision-based indexing (the fastest configuration we tested), with varying query selectivity. By query selectivity, we refer to the percentage of the column data returned by a query. For the GTS potential column, we provide a speedup in the range of 3.2 to 11.9. For the S3D temperature column, a speedup of 5.2 to 9.0 is observed. Due to the clustering of the data, a very small number of I/O seek operations are needed by our method relative to FASTBIT. Furthermore, the amount of data read by our method is much lower than that by FASTBIT, as shown in Table 1. The reason that sequential scan performs better than FASTBIT in this context is that, in parallel file systems such as Lustre, seeks are a very high-latency operation. For value-centric queries, FASTBIT incurs a seek per item, whereas sequential scan reads all data in a single, large read, and so for less selective queries, the seek costs outweigh the read costs.

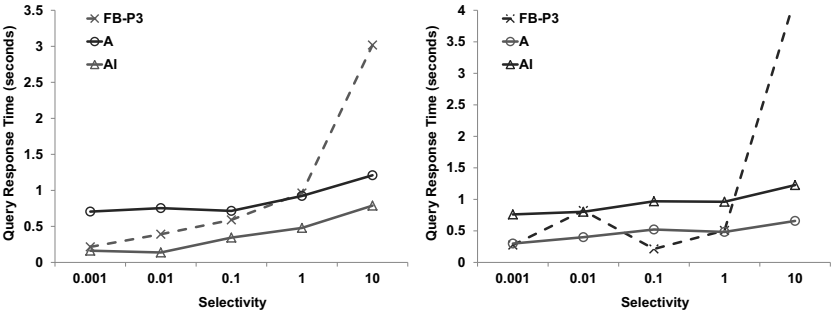


Fig. 9. Comparison of response return by FASTBIT (FB-P3) against ALACRITY, with and without inverted index compression (ALAC_I and ALAC, respsectively), for region-centric queries with varying number of query hits. The left and right plot show results for S3D temperature and GTS potential, respectively.

Region-centric Queries. Figure 9 shows region query response time with varying number of hits (records returned) for our method compared to FASTBIT with precision and default binning. As mentioned, the queries were chosen so that no candidate checks are needed. The precision-based boundaries are the basis for our method and is base two, while for FASTBIT it is a special configuration using base ten. Overall, it is seen that query performance time is low in nearly all cases on account of only needing to process the index, but different configurations of ALACRITY are able to meet or slightly outperform the FASTBIT alternative, likely due to the lower index size, though decompression overhead is a concern, as shown in the right plot of Figure 9. For the GTS potential variable, the compressed index size for $k = 3$ was less space-efficient than that for $k = 2$, making the read+decompress overhead larger than just reading the raw inverted index.

4.4 Performance Analysis

Figure 10 shows the breakup of overall query processing time into I/O and compute components, corresponding to index/bin loading and processing, respectively. The dataset tested on is S3D using the velocity variable. I/O is the dominant cost of query processing, while the application of the query constraints and data transformations is a low, though non-negligible, component. We believe multithreading or asynchronous I/O would be able to hide most of the compute costs by interleaving it with the more costly I/O operations.

4.5 Compression

To analyze the performance of our lossless data compression scheme, we compare the compression ratios obtained with our method (without the inverted index) to those obtained by general-purpose lossless compression utilities, as well as more recent floating-point compressors. Out of the datasets tested, our method performed better than all of

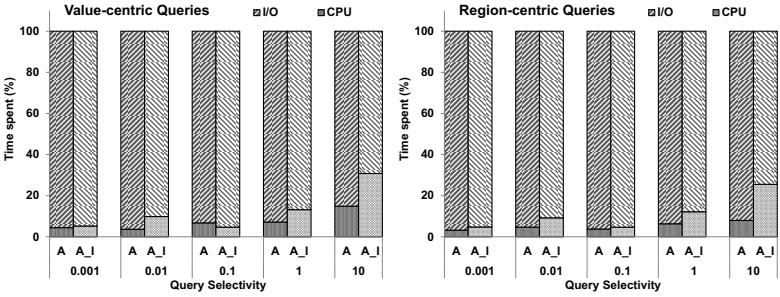


Fig. 10. Comparison of computation and I/O time distribution for ALACRITY for different query types of varying selectivity, on the S3D temperature variable. A: ALACRITY without bin or inverted index compression. A_I: ALACRITY with inverted index compression.

Table 4. Compression ratio and ALACRITY storage components. A_B: ALACRITY with bin compression (using bzip2).

Dataset	Compression Ratio					Storage Requirement (%)		
	gzip	fpzip	bzip2	FPC	A_B	Data	Index	Metadata
msg_bt	1.12	1.20	1.09	1.29	**1.40**	69.35	1.86	≈0.00
msg_lu	1.05	1.13	1.01	1.17	**1.30**	74.42	1.97	0.01
msg_sp	1.10	1.11	1.06	1.26	**1.33**	73.98	1.11	≈0.00
msg_sppm	7.41	3.25	7.09	5.30	**8.87**	9.58	1.66	0.02
msg_sweep3d	1.09	1.33	1.32	**3.09**	2.11	46.60	0.67	0.02
num_brain	1.06	1.25	1.06	1.16	**1.28**	74.50	3.39	≈0.00
num_comet	1.16	1.27	1.17	1.16	**1.34**	66.16	8.16	0.03
num_control	1.05	1.12	1.03	1.05	**1.15**	74.02	12.22	0.02
num_plasma	1.77	1.06	6.17	15.05	**75.72**	0.70	0.60	0.03
obs_error	1.44	1.37	1.36	**3.60**	2.59	34.07	4.51	≈0.00
obs_info	1.14	1.06	1.22	2.27	**3.52**	24.97	3.36	0.04
obs_spitzer	1.23	1.07	1.78	1.03	**1.90**	44.36	8.05	≈0.00
obs_temp	1.03	1.09	1.03	1.02	**1.13**	75.00	12.70	0.03
gts_phi_l	1.04	1.18	1.02	1.07	**1.19**	75.00	8.56	0.03
gts_phi_nl	1.04	1.17	1.01	1.07	**1.19**	75.00	9.20	0.03
gts_chkp_zeon	1.04	1.09	1.02	1.01	**1.17**	75.00	10.04	0.10
gts_chkp_zion	1.04	1.10	1.02	1.02	**1.18**	75.00	9.60	0.11
gts_potential	1.04	1.15	1.01	1.06	**1.18**	75.00	9.60	≈0.00
xgc_iphase	1.36	1.53	1.37	1.36	**1.58**	55.33	7.56	≈0.00
s3d_temp	1.18	**1.46**	1.15	1.34	1.35	73.38	0.77	≈0.00
s3d_vvel	1.04	1.24	1.02	1.15	**1.27**	75.00	3.74	≈0.00
flash_velx	1.11	**1.34**	1.08	1.26	1.32	75.00	0.81	≈0.00
flash_vely	1.13	**1.43**	1.09	1.29	1.32	75.00	0.80	≈0.00
flash_gamc	1.28	**1.62**	1.28	1.53	1.40	71.37	0.06	≈0.00

the other compressors tested (gzip, fpzip [17], bzip2, and FPC [7]) on 18 of 24. FPC gave superior performance compared to our method on two of the 27 datasets, while fpzip gave better performance on the remaining four. Overall, our method was consistent in yielding comparable or better compression ratios than the other compressors, providing evidence of strong compression ratios in other application datasets.

To justify our superior performance on most of the datasets, we argue that the bin-based compression of the data generally allows a much greater exploitation of existing compression algorithms than the normal distribution of scientific data that was passed to the other compressors. The reorganization of the data allowed bzip2 to be utilized as best as possible, causing the data to be reduced significantly because of the splitting of the low-entropy and high-entropy sections of the data. As evidenced by the small compressed index and metadata sizes, the reorganization is a low-overhead operation with respect to storage. We attribute the better performance of FPC and fpzip on some of the datasets to the encoding of data dependency which the FCM [28], DFCM [10], and Lorenzo [12] predictors used by FPC and fpzip were able to capture in their predictions.

5 Conclusion

As the size of scientific datasets in various disciplines continues to grow, new methods to store and analyze the datasets must be developed, as I/O capabilities are not growing as quickly, and new technologies (such as SSDs) are not currently able to achieve the storage density and cost-efficiency of traditional mechanical disk drives. Successful methods of mitigating this growing gap must involve data reduction in all stages of the knowledge discovery pipeline, including storage of raw data as well as analytics metadata. We believe our work in this paper in compression, indexing, and query processing of scientific data represents a step in the right direction, allowing both efficient lossless compression of floating-point data for accuracy-sensitive applications as well as efficient query processing on variable constraints, all with less space and I/O requirements than other database technologies.

Acknowledgements. We would like to acknowledge the use of resources at ORNL's leadership class computing facility, OLCF. Also, we appreciate the use of the datasets available from the Flash Center for Computational Science. This work was supported in part by the U.S. Department of Energy, Office of Science and the U.S. National Science Foundation (Expeditions in Computing and EAGER programs). Oak Ridge National Laboratory is managed by UT- Battelle for the LLC U.S. D.O.E. under contract no. DEAC05- 00OR22725.

References

1. IEEE standard for floating-point arithmetic. IEEE Standard 754-2008 (2008)
2. Abadi, D., Madden, S., Ferreira, M.: Integrating compression and execution in column-oriented database systems. In: Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data, SIGMOD 2006, pp. 671–682. ACM, New York (2006)

3. Anh, V.N., Moffat, A.: Index compression using fixed binary codewords. In: Proceedings of the 15th Australasian Database Conference, ADC 2004, vol. 27, pp. 61–67. Australian Computer Society, Inc., Darlinghurst (2004)
4. Antoshenkov, G.: Byte-aligned bitmap compression. In: Data Compression Conference, p. 476 (1995)
5. Fryxell, B., Olson, K., Ricker, P., Timmes, F.X., Zingale, M., Lamb, D.Q., MacNeice, P., Rosner, R., Truran, J.W., Tufo, H.: FLASH: An adaptive mesh hydrodynamics code for modeling astrophysical thermonuclear flashes. The Astrophysical Journal Supplement Series 131, 273–334 (2000)
6. Burtscher, M., Ratanaworabhan, P.: High throughput compression of double-precision floating-point data. In: IEEE Data Compression Conference, pp. 293–302 (2007)
7. Burtscher, M., Ratanaworabhan, P.: FPC: A high-speed compressor for double-precision floating-point data. IEEE Transactions on Computers 58, 18–31 (2009)
8. Chen, J.H., Choudhary, A., Supinski, B., DeVries, M., Hawkes, E.R., Klasky, S., Liao, W., Ma, K., Mellor-Crummey, J., Podhorszki, N., Sankaran, R., Shende, S., Yoo, C.: Terascale direct numerical simulations of turbulent combustion using S3D. Comp. Sci. and Discovery 2(1)
9. Comer, D.: The ubiquitous B-Tree. ACM Comput. Surv. 11, 121–137 (1979)
10. Goeman, B., Vandierendonck, H., Bosschere, K.D.: Differential FCM: Increasing value prediction accuracy by improving table usage efficiency. In: Seventh International Symposium on High Performance Computer Architecture, pp. 207–216 (2001)
11. Graefe, G., Shapiro, L.: Data compression and database performance. In: Proceedings of the 1991 Symposium on Applied Computing, pp. 22–27 (April 1991)
12. Ibarria, L., Lindstrom, P., Rossignac, J., Szymczak, A.: Out-of-core compression and decompression of large n-dimensional scalar fields. Computer Graphics Forum 22, 343–348 (2003)
13. Isenburg, M., Lindstrom, P., Snoeyink, J.: Lossless compression of predicted floating-point geometry. Computer-Aided Design 37(8), 869–877 (2005); CAD 2004 Special Issue: Modelling and Geometry Representations for CAD
14. Iyer, B.R., Wilhite, D.: Data compression support in databases. In: Proceedings of the 20th International Conference on Very Large Data Bases, VLDB 1994, pp. 695–704. Morgan Kaufmann Publishers Inc., San Francisco (1994)
15. Jenkins, J., et al.: Analytics-driven lossless data compression for rapid in-situ indexing, storing, and querying. In: Liddle, S.W., Schewe, K.-D., Tjoa, A.M., Zhou, X. (eds.) DEXA 2012, Part II. LNCS, vol. 7447, pp. 16–30. Springer, Heidelberg (2012)
16. Ku, S., Chang, C., Diamond, P.: Full-f gyrokinetic particle simulation of centrally heated global ITG turbulence from magnetic axis to edge pedestal top in a realistic Tokamak geometry. Nuclear Fusion 49(11), 115021 (2009)
17. Lindstrom, P., Isenburg, M.: Fast and efficient compression of floating-point data. IEEE Transactions on Visualization and Computer Graphics 12, 1245–1250 (2006)
18. Schendel, E.R., Jin, Y., Shah, N., Chen, J., Chang, C., Ku, S.-H., Ethier, S., Klasky, S., Latham, R., Ross, R., Samatova, N.F.: ISOBAR preconditioner for effective and high-throughput lossless data compression. In: Proceedings of the 28th International Conference on Data Engineering, ICDE 2012. IEEE (2012)
19. Sinha, R.R., Winslett, M.: Multi-resolution bitmap indexes for scientific data. ACM Trans. Database Syst. 32 (2007)
20. Wang, W.X., Lin, Z., Tang, W.M., Lee, W.W., Ethier, S., Lewandowski, J.L.V., Rewoldt, G., Hahm, T.S., Manickam, J.: Gyro-kinetic simulation of global turbulent transport properties in Tokamak experiments. Physics of Plasmas 13(9), 092505 (2006)
21. Westmann, T., Kossmann, D., Helmer, S., Moerkotte, G.: The implementation and performance of compressed databases. SIGMOD Rec. 29(3), 55–67 (2000)

22. Witten, I.H., Moffat, A., Bell, T.C.: Managing Gigabytes: Compressing and Indexing Documents and Images, 2nd edn. Morgan Kaufmann (1999)
23. Wu, K.: Fastbit: an efficient indexing technology for accelerating data-intensive science. Journal of Physics: Conference Series 16, 556 (2005)
24. Wu, K., Ahern, S., Bethel, E.W., Chen, J., Childs, H., Cormier-Michel, E., Geddes, C., Gu, J., Hagen, H., Hamann, B., Koegler, W., Lauret, J., Meredith, J., Messmer, P., Otoo, E., Perevoztchikov, V., Poskanzer, A., Prabhat, Rubel, O., Shoshani, A., Sim, A., Stockinger, K., Weber, G., Zhang, W.-M.: FastBit: interactively searching massive data. Journal of Physics: Conference Series 180(1), 012053 (2009)
25. Wu, K., Otoo, E., Shoshani, A.: On the performance of bitmap indices for high cardinality attributes. In: Proc. of the Thirtieth International Conference on Very Large Data Bases, VLDB 2004, vol. 30, pp. 24–35 (2004)
26. Wu, K., Otoo, E.J., Shoshani, A.: Optimizing bitmap indices with efficient compression. ACM Trans. Database Syst. 31, 1–38 (2006)
27. Yan, H., Ding, S., Suel, T.: Inverted index compression and query processing with optimized document ordering. In: Proceedings of the 18th International Conference on World Wide Web, WWW 2009, pp. 401–410. ACM, New York (2009)
28. Yiannakis, S., Smith, J.E.: The predictability of data values. In: Proceedings of the 30th Annual ACM/IEEE International Symposium on Microarchitecture, MICRO 30, pp. 248–258. IEEE Computer Society, Washington, DC (1997)
29. Zobel, J., Moffat, A.: Inverted files for text search engines. ACM Computing Surveys 38(2) (July 2006)
30. Zukowski, M., Heman, S., Nes, N., Boncz, P.: Super-scalar ram-cpu cache compression. In: Proceedings of the 22nd International Conference on Data Engineering, ICDE 2006, pp. 59–71. IEEE Computer Society, Washington, DC (2006)

A Declarative Approach to View Selection Modeling

Imene Mami, Zohra Bellahsene, and Remi Coletta

University Montpellier 2, LIRMM, France
{mami,bella,coletta}@lirmm.fr

Abstract. View selection is important in many data-intensive systems e.g., commercial database and data warehousing systems. Given a database (or a data warehouse) schema and a query workload, view selection is to choose an appropriate set of views to be materialized that optimizes the total query cost, given a limited amount of resource, e.g., storage space and total view maintenance cost. The view selection problem is known to be a NP-complete problem. In this paper, we propose a declarative approach that involves a constraint programming technique which is known to be efficient for the resolution of NP-complete problems. The originality of our approach is that it provides a clear separation between formulation and resolution of the problem. For this purpose, the view selection problem is modeled as a constraint satisfaction problem in an easy and declarative way. Then, its resolution is performed automatically by the constraint solver. Furthermore, our approach is flexible and extensible, in that it can easily model and handle new constraints and new heuristic search strategies to reduce the solution space. The performance results show that our approach outperforms the genetic algorithm which is known to provide the best trade-off between quality of solutions in terms of cost saving and execution time.

keywords: Database design, modeling and management, query processing and optimization, view selection, materialized views.

1 Introduction

Selecting the best set of views to materialize for a given query workload, under certain resource constraints, is one of the most common problems in commercial database management systems and data warehousing systems. In many applications, and in particular in a data warehouse application, queries need to be answered over massive amounts of data. Materializing and exploiting previous query results (views) can be important for efficient processing of queries by avoiding re-computation of expensive query operations. Consequently, answering queries using materialized views is significant for improving query performance. To support view selection process, different related issues have to be considered. One of the challenging issues is the view maintenance which is the process of updating a materialized view. Indeed, whenever a data source is changed, the materialized views built on it have to be updated (or at least have to be checked

A. Hameurlain et al. (Eds.): TLDKS X, LNCS 8220, pp. 115–145, 2013.

whether some changes have to be propagated or not) in order to compute up-to-date query results. The view maintenance cost constraint is very important in the view selection process and cannot be ignored. Otherwise, the cost of the view maintenance may offset the performance advantages provided by the view materialization. Besides the view maintenance issue, each materialized view requires additional storage space which must be taken into account when deciding which and how many views to materialize. Hence, there is a need for selecting a set of views to materialize by taking into account three important features: query cost, view maintenance cost and storage space. The problem of choosing which views to materialize that minimize the total query cost given a limited amount of resource such as total view maintenance cost and storage space is known as the view selection problem. This is one of the most complex problem solving: it is known to be a NP-complete problem [11]. Moreover, the number of possible view combinations to materialize grows exponentially with the number of queries and with the numbers of columns, join predicates and grouping clauses.

There has been much work on materialized view selection. A naive method is to apply an algorithm which finds the optimal set of materialized views by browsing through all sets of considered views to materialization. However, an exhaustive search cannot be applied due to the complexity of the problem. The most efficient method for deciding which views to be materialized for a given workload is a randomized method which uses the genetic algorithm [5,18,33]. The main difference between the genetic algorithm and previously designed algorithms, i.e., greedy algorithms [7,8,24,28,32] is that it can be applicable on the large search space. It can find a reasonable solution within a relatively short period of time by trading executing time for quality. However, there is no guarantee of performance because the probabilistic behavior of the genetic algorithms does not insure to find the global optimum. Besides, the quality of the solution (i.e., the quality of the obtained set of materialized views in terms of cost saving) depends on the set-up of the algorithm as well as the extremely difficult fine-tuning of the algorithm that must be performed during many test runs.

In this paper, we have proposed a constraint programming based approach. Constraint programming is a general framework which relies on a combination of techniques that deal with reasoning. It has been applied with success to many domains such as scheduling, planning, vehicle routing, configuration, networks and bioinformatics. More recently, constraint programming has been considered as beneficial in data mining setting [25]. Our motivation to use constraint programming in solving the view selection problem is that it is known to be efficient for the resolution of NP-complete problems and a powerful method for modeling and solving combinatorial optimization problems [26]. To solve a given problem by means of constraint programming, the problem must be represented as a constraint satisfaction problem. This part of the problem solving is called modeling. Then, the resolution of the modeled problem is performed automatically by the constraint solver in the solving stage. The originality of our approach is that it provides a clear separation between formulation and resolution of the problem. Indeed, constraint programming is a declarative programming paradigm: instead

of specifying how to solve the problem, the user has only to specify the problem itself.

Our Goals. Based on the application workload, we select a set of views to materialize over a database (or data warehouse) schema, such that the cost of evaluating queries is minimal, subject to space and maintenance cost constraints. Our goal is to provide better solution quality (i.e., the quality of the obtained set of materialized views in terms of cost saving) with respect to the currently most efficient approach (genetic algorithm). The focus of this study is also to enable optimal view selection by using constraint programming techniques.

Our Contributions. We propose a novel and efficient approach to address the view selection problem. Our approach is based on constraint programming techniques and consists in modeling in a declarative way the view selection as a constraint satisfaction problem. We formalize and study the view selection problem under a limited amount resource, e.g., storage space and view maintenance cost. The contributions of this paper are based on the extension of our previous work [23].

- We include further explanations and illustrations through the paper, i.e., how the constraint programming can be applied to decide which views to materialize (see section 3.2).
- We propose a heuristic search strategy to efficiently search the solution space (see section 5.2.2). We prove that the time that a constraint solver incurs for finding near optimal and optimal solutions is significantly reduced (see section 6.2).
- We also show the effectiveness of our heuristic based search strategy which improves in several magnitudes the quality of the solution provided by the previous version [23]. Hence, our approach achieves significant performance gains compared with the genetic algorithm in terms of cost saving (see section 6.3.1 and 6.3.2). While in our previous work [23] we achieved only a slight improvement.
- We design new and various experiments to prove the efficiency of our approach when we simulate diverse query workloads by generating different query and update distribution and query complexity. The results show that, over all the experiments, the performance of our approach is much better than that of the genetic algorithm (see section 6.3.3 and 6.3.4).
- We perform real experiments on MySQL server in order to measure the *real* query runtime (see section 6.4). The results of these experiments have shown that queries using our proposed views are evaluated faster in comparison with those found by the genetic algorithm. These experiments also confirm the robustness of our approach toward simplified cost models. This requirement is very important for database optimization as it is based on cost estimations.

Paper Outline. The rest of this paper is organized as follows. After reviewing and classifying prior work in the view selection context, Section 3 contains the background related to understand the view selection problem and discusses the settings for the problem. In Section 4, we present the framework that we have used for representing views to materialize in order to exhibit common subexpressions. Section 5 describes how to model the view selection problem as a

constraint satisfaction problem as well as the heuristic search strategy that we have designed for optimization purpose. Section 6 gives a performance analysis comparing our approach with the genetic algorithm. The paper ends with a summary and future works in Section 7.

2 Related Work

In this section, we review the view selection methods based on what kind of algorithms they use to address the view selection problem. The best-known heuristic algorithms proposed in literature to tackle the problem of finding an appropriate set of views to materialize can be classified into three major groups: deterministic algorithms, randomized algorithms and hybrid algorithms. For a deeper review of the existing view selection approaches, we refer the reader to the survey that we have done in our previous work [21].

Deterministic Algorithms Based Methods. Much research work on view selection uses deterministic strategies to address the view selection problem. [27] is the first paper that provides a solution for materializing view indexes which can be seen as a special case of the materialized views. The solution is based on A* algorithm. An exhaustive approach is also presented in [16] for finding the best set of views to materialize. Nevertheless, an exhaustive search cannot compute the optimal solution in a reasonable time.

The authors in [9] present and analyze algorithms for view selection in case of OLAP-style queries. They provide a polynomial-time greedy algorithm to select a set of views to materialize that minimizes the query cost subject to a space constraint. However, this approach does not consider the view maintenance cost. The work in [32] is dealing with more general SQL queries which include select, project, join, and aggregation operations. A greedy algorithm has been designed to select a set of materialized views so that the combined query and view maintenance cost is minimized. However, the view maintenance cost has been overrated since the maintenance cost for a materialized view is the cost used for constructing this view. Besides, the view selection is done without any resource constraint.

A theoretical framework for the view selection problem in data warehousing setting has been developed in [7]. Their work provides a near-optimal polynomial time greedy algorithm for the cases of AND view graph, where each query (or view) has a unique evaluation, and OR view graph, in which any view can be computed from any one of its related views. For the most general case of AND-OR view graph which allows a single query to be answered and updated from multiple paths, they have designed a near-optimal exponential time greedy algorithm. This approach was extended in [8] to study the view selection under a maintenance cost constraint.

The view selection has been studied in [19,30,31] under the condition that the input queries can be answered using exclusively the materialized views. An exhaustive algorithm has been designed in [31] to select a set of materialized views while minimizing the combination of the query and view maintenance cost. This work was extended in [19] by developing greedy algorithms that expand only

a small fraction of the states produced by the exhaustive algorithm. The view selection problem in [30] is addressed under a space constraint. However, their view selection algorithm is still in exponential time.

The system designed in [3] runs a greedy enumeration algorithm to pick a set of views and indexes to materialize by taking into account the space constraint. Nevertheless, this approach does not take into account the view maintenance cost.

The authors in [28] demonstrate that using multi-query optimization techniques in conjunction with a greedy heuristic provides significant benefit. The greedy heuristic is used to iteratively pick from the AND-OR view graph the set of views to materialize that minimizes the query cost. This study was extended in [24] to consider how to optimize the view maintenance cost. However, the view selection has been studied without any resource constraint.

In order to improve the query performance as well as save the storage space, the study in [29] aims at materializing only a part of the relations instead of considering all tuples in the relations. An efficient algorithm has been designed which uses clustering techniques to select the set of views to be materialized.

The above methods take a deterministic approach either by exhaustive search or by some heuristics such as greedy. However, greedy search is subjected to the known caveats, i.e., sub-optimal solutions may be retained instead of the globally optimal one since initial solutions influence the solution greatly. As a result, other algorithms have been developed to improve the solutions of the view selection problem, namely: randomized algorithms and hybrid algorithms which we describe in next sections.

Randomized Algorithms Based Methods. Typical randomized algorithms are genetic or use simulated annealing. Genetic algorithms generate solutions using techniques inspired by the natural evolution process such as selection, mutation, and crossover. The search strategy for these algorithms is very similar to biological evolution. Genetic algorithms start with a random initial population and generate new populations by random crossover and mutation. The fittest individual found is the solution. The algorithms terminate as soon as there is no further improvement over a period.

A genetic algorithm has been used in [33] to solve the view selection problem. The materialized views have been selected according to their reduction in the combined query and view maintenance cost. However, because of the random characteristic of the genetic algorithm, some solutions can be infeasible. For example, in the maintenance cost constrained model, when a view is selected, the benefit will not only depend on the view itself but also on other views that are selected. One solution to this problem is to add a penalty value as part of the fitness function to ensure that infeasible solutions will be discarded. For instance, a penalty function has been applied in [18] which reduces the fitness each time the maintenance cost constraint is not satisfied. This approach minimizes the query cost given varying upper bounds on the view maintenance cost, assuming unlimited amount of storage space. In order to let the genetic algorithm converge faster, they represent the initial population as a favorable configuration based

on external knowledge about the problem and its solution rather than a random sampling, i.e., the views with a high query frequency are most likely selected for materialization.

The approach proposed in [10] use simulated annealing algorithms to address the view selection problem. These algorithms are motivated by an analogy to annealing in solids. Simulated Annealing algorithms start with an initial configuration, generate new configurations by random walk along the different solutions of the solution space according to a cooling schedule and terminate as soon as no applicable ones exist or lose all the energy in the system. The view selection problem is solved in [10] under the case where either the space constraint or the maintenance cost constraint is considered. Further, randomized search has been applied to solve two more issues. First, they considered the case where both space and maintenance constraints exist. Next they applied a randomized search in the context of dynamic view selection.

In contrast with simulated annealing algorithms, genetic algorithms use a multi-directional search which allows to efficiently search the space and find better solution quality. For more details about this observation we refer the reader to [18]. Randomized algorithms can be applied to complex problems dealing with large or even unlimited search spaces. Recent works [13,14] have shown that randomized search heuristic techniques, in comparison to greedy techniques, are able to select comparatively better quality views for higher dimensional data sets. However, they may have a tendency to converge toward local optima due to their random characteristics. Besides, their successes often depend on the set-up of the algorithm as well as the extremely difficult fine-tuning of algorithm that must be performed during many test runs.

Hybrid Algorithms Based Methods. Hybrid algorithms combine the strategies of deterministic and randomized algorithms in their search in order to provide better performance in terms of solution quality. Solutions obtained by deterministic algorithms are used as initial configuration for simulated annealing algorithms or as initial population for genetic algorithms.

A hybrid approach has been applied in [34] which combines heuristic algorithms i.e., greedy algorithms and genetic algorithms to solve three related problems. The first one is to optimize queries. The second one is to choose the best global processing plan from multiple processing plans for each query. The third problem is to select materialized views from a given global processing plan. Their experimental results confirmed that hybrid algorithms provide better performance than either genetic algorithms or heuristic algorithms i.e., greedy algorithms used alone in terms of solution quality. However, their algorithms are more time consuming and may be impractical due to their excessive computation time.

3 Background

3.1 View Selection Problem and Cost Model

View Selection Problem The problem of view selection that we consider in this paper is to select a set of views to be materialized in order to speed up a given set

of queries constrained by a storage space capacity and maintenance costs to keep the materialized views in synchronization with the underlying base relations.

More precisely, the view selection problem can be defined as follows: *Given a query workload* $Q = \{q_1, q_2, ..., q_q\}$ *and their query frequency* $fQ = \{f_{q_1}, f_{q_2}, ..., f_{q_q}\}$ *over a given database (or data warehouse) schema* $R = \{r_1, r_2, ..., r_r\}$*, a set of updates* $U = \{u_1, u_2, ..., u_u\}$ *on base relations and their update frequency* $fU = \{fu_{r_1}, fu_{r_2}, ..., fu_{r_r}\}$ *and a limited amount of resource, e.g., storage space* Sp_{max} *and view maintenance cost limit* U_{max}*, the problem is to find a set of views to materialize* $MV = \{v_1, v_2, ..., v_v\}$ *such as the cost of evaluating the query workload is minimal.*

Cost Model The cost model assigns an estimated cost e.g., query cost or maintenance cost to any view (or query) in the search space. In our approach, we use a cost model similar to [6,20]. Hence, the query and view maintenance costs are estimated with respect to CPU and IO costs. In this paper we consider selection-projection-join (SPJ) queries that may involve aggregation and a group by clause as well. The formulas used for cost operations estimation are given below with the following assumptions:

- Formulas to estimate the cost of executing every relational operation take into account its implementation, e.g., we consider sequential scans and nested loop joins.
- The CPU cost is estimated as the time needed to process each tuple of the relation e.g., checking selection conditions.
- The IO cost estimation is the time necessary for fetching each tuple of the relation.
- The costs are estimated according to the size of the involved relations and in terms of time.

Estimated Cost of Relational Operations

- Estimated cost of unary operations
  - $cost(op) = (IO * card * length) + (CPU * card * lengthP)$ where op is a selection operation
  - $cost(op) = (IO * card * log(card) * length) + (CPU * card * log(card) * lengthP)$ where op is a projection operation
  - $cost(op) = (IO * card * length) + (CPU * card * lengthA)$ where op is an aggregation operation
- Estimated cost of binary operations
  - $cost(op) = (IO * lcard * rcard * (llength + rlength)) + (CPU * lcard * rcard * lengthP)$ where op is a join operation

Where *card* is the number of tuples of the operand, *length* is the length (in bytes) of a tuple, *lengthP* is the length of columns checked by predicates, *lengthA* is the length of the tuples being aggregated, *lcard* and *rcard* are respectively the number of tuples of the left and right operands (the same for *llength* and *rlength*).

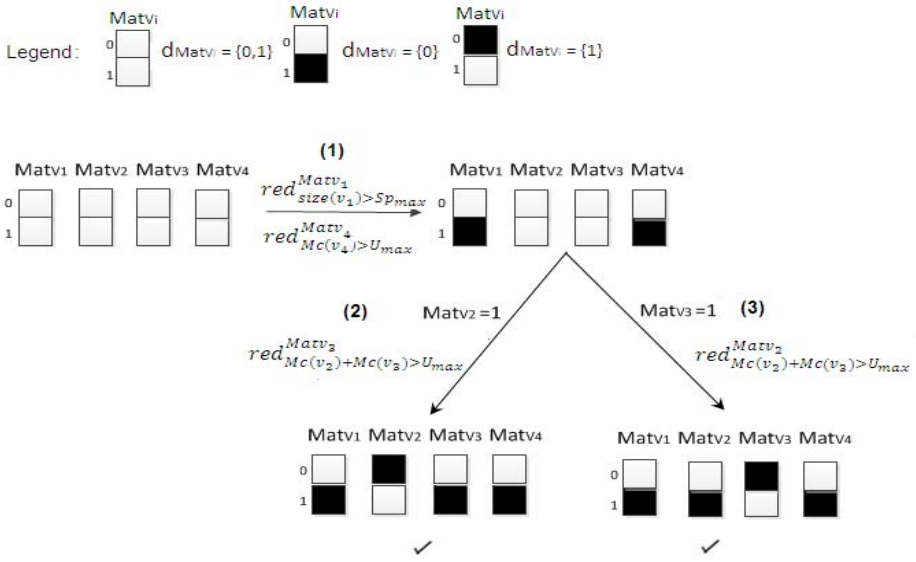


Fig. 1. Search tree using constraint propagation

3.2 Constraint Programming Notions

Constraint programming has been successfully applied in numerous combinatorial search problems [26] such as scheduling and timetabling. Constraint programming allows to solve combinatorial problems modeled as a Constraint Satisfaction Problem (CSP). Indeed, the principle idea of constraint programming is to solve problems by stating constraints which must be satisfied by the solution.

Formally, a CSP is defined by a triplet (VAR;DOM;CST):

- Variables. $VAR = \{var_1, var_2, ..., var_n\}$ is the set of variables of the problem.
- Domains. $DOM = \{d_{var_1}, d_{var_2}, ..., d_{var_n}\}$ is the set of possible values that can be assigned to each variable var_i.
- Constraints. $CST = \{c_1, c_2, ..., c_n\}$ is the set of constraints that describes the relationship between subsets of variables. Formally, a constraint C_{ijk} between the variables var_i, var_j, var_k is any subset of the possible combinations of values of var_i, var_j, var_k, i.e., $C_{ijk} \subset d_{var_i} \times d_{var_j} \times d_{var_k}$. The subset specifies the combinations of values that the constraint allows.

A feasible solution to a CSP is an assignment of a value from its domain to every variable, so that the constraints on these variables are satisfied. For optimization purpose some cost expression on these variables takes a maximal or minimal value.

Most algorithms for solving CSPs usually use constraint propagation to reduce the size of the search space to be explored [15]. When a value of a variable is fixed, constraint propagation is applied to restrict the domains of other variables whose values are not currently fixed. This means that when a value is assigned to

the current variable, any value in the domain of a future variable which conflicts with this assignment is removed from the domain.

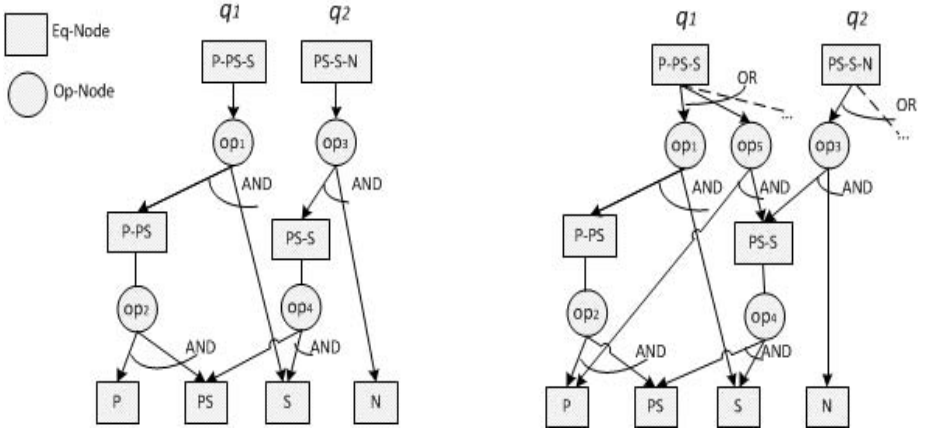
Let us now illustrate this in the context of view selection problem. Figure 1 shows the domain reduction of four variables Mat_{v_1}, Mat_{v_2}, Mat_{v_3} and Mat_{v_4} where Mat_{v_i} denotes for each view v_i if it has been materialized or has not been materialized. It is a binary variable, $d_{Mat_{v_i}} = \{0,1\}$ (0: v_i has not been materialized, 1: v_i has been materialized). The problem is to select a set of views to materialize subject to a space and maintenance cost constraints. The space constraint ensures that the total space occupied by the materialized views is less than Sp_{max}. Let as assume that Sp_{max}=3MB, $size(v_1)$=4MB, $size(v_2)$=2MB, $size(v_3)$=1MB and $size(v_4)$=1MB; where $size(v_i)$ is the size of the view v_i. While, the maintenance cost constraint guarantees that the time to update the set of materialized views is less than U_{max}. Note that $U_{max} = 3sec$, $Mc(v_1)$=1sec, $Mc(v_2)$=2sec, $Mc(v_3)$=2sec and $Mc(v_4)$=5sec; where $Mc(v_i)$ denotes the cost of maintaining the view v_i.

At the beginning, the initial variable domains, $d_{Mat_{v_1}} = d_{Mat_{v_2}} = d_{Mat_{v_3}} = d_{Mat_{v_4}} = \{0,1\}$, are represented by four columns of white squares. Considering the space and maintenance cost constraints, it appears that Mat_{v_1} and Mat_{v_4} cannot take the value 1 because otherwise the total space and maintenance cost of the materialized views will be respectively greater than Sp_{max} and U_{max}. In the stage (1), $red^{Mat_{v_1}}_{size(v_1)>Sp_{max}}$ and $red^{Mat_{v_4}}_{Mc(v_4)>U_{max}}$ filters respectively the inconsistent value 1 from $d_{Mat_{v_1}}$ and $d_{Mat_{v_4}}$. The deleted values are marked with a black square. After this stage some variable domains are not reduced to singletons, the constraint solver takes one of these variables and tries to assign to it each of the possible values in turn. For example, if the solver selects the view v_2 to be materialized ($Mat_{v_2} = 1$, see stage (2)), $red^{Mat_{v_3}}_{Mc(v_2)+Mc(v_3)>U_{max}}$ eliminates the value 1 from $d_{Mat_{v_3}}$. Otherwise, if the view v_3 is selected to be materialized ($Mat_{v_3} = 1$, see stage (3)), $red^{Mat_{v_2}}_{Mc(v_2)+Mc(v_3)>U_{max}}$ withdraws the value 1 from $d_{Mat_{v_2}}$. This enumeration stage leads in our example to two solutions. These solutions are of various quality or cost.

In addition to providing a rich constraint language to model a problem as a CSP and techniques such as constraint propagation to reduce the search space by excluding solutions where the constraints become inconsistent, constraint programming offers facilities to control the search behavior. This means that search strategies can be defined to decide in which order to explore the created child nodes in an enumeration tree which can significantly reduce the execution time. Furthermore, constraint programming provides ways to limit the tree search regarding different criteria. For instance performing the search until reaching a feasible solution in which all constraints are satisfied, or until reaching a search time limit or until reaching the optimal solution.

4 Framework for Detecting Common Views

In our approach, the task of a view selection module is to recognize possibilities of shared views and then to apply a strategy that use constraint programming



(a) AND view graph (b) AND-OR view graph

Fig. 2. DAG representation of two queries q_1 and q_2

techniques for deciding which views to materialize. The first task involves setting up the search space by identifying common sub-expressions between the different queries of workload. This feature can be exploited for sharing computation, updates and storage space. The most commonly used frameworks in the context of representing SQL queries in order to exhibit common sub-expressions are the AND view graph and the AND-OR view graph. In what follows, we start by giving a formal definition of these representations.

Definition 4.1 (AND View Graph) *An AND view graph is formed from the union of individual AND-DAG representations of each query. An AND-DAG representation for a query or a view v is a directed acyclic graph having the base relations as leaf nodes and the node v as a root node and consists of a set of operation nodes (Op-Nodes) and equivalence nodes (Eq-Nodes). The Op-nodes have only Eq-nodes as children and Eq-nodes have only Op-nodes as children. Each Op-Node corresponds to an algebraic expression (Select-Project-Join) with possible aggregate function. It represents the expression defined by the operand and its inputs. An Eq-Node represents an expression that is defined by the child operation node and its inputs. Each Eq-Node represents a view that could be selected for materialization. In an AND-DAG representations, each Op-node* op_i *has associated with it an AND arc which is indicated by drawing a semicircle, through the edges* (op_i,v_{c_1}),(op_i,v_{c_2}),...,(op_i,v_{c_i}). *This dependence means that all the views* v_{c_1}, v_{c_2},...,v_{c_i} *that are the child nodes of* op_i *are needed to compute the view* v_p *which is the parent node of* op_i.

Definition 4.2 (AND-OR View Graph) *A graph is called an AND-OR view graph if for each query or a view v, there is an AND-OR-DAG representation. All the possible AND-DAG representations for v, described in the previous definition,*

become the AND-OR DAG which consists of all possible execution plans for v. If a parent view v_p has outgoing edges to children operation nodes op_1,op_1,...,op_i, then v_p can be computed from any one of its children. This dependence is indicated by drawing a semicircle, called an OR arc. The AND-OR view graph can be constructed by merging the AND-OR DAG for each query where the common sub-expressions are represented once.

The DAG representation of the queries q_1: P $\bowtie$ PS $\bowtie$ S and q_2: PS $\bowtie$ S $\bowtie$ N, are shown in figure 2. The subscripts P, PS, S and N denote respectively the base relations of TPC-H benchmark: Part, PartSupp, Supplier and Nation. In the AND view graph (see figure 2a), there is only one way to answer or update a view (or query). Indeed, the views P-PS-S and PS-S-N, corresponding respectively to the result of the query q_1 and q_2, can be computed or updated on only one way (it consider optimal query plans):

q_1**:((P $\bowtie$ PS) $\bowtie$ S)**
q_2**:((PS $\bowtie$ S) $\bowtie$ N)**

However, all possible ways for evaluating the queries have been considered in the AND-OR view graph (see figure 2b). For simplicity, we represent only two execution plans for the view P-PS-S which is the query result of q_1 and one execution plan for the view PS-S-N that is the query result of q_2:

q_1**:{((P $\bowtie$ PS) $\bowtie$ S), (P $\bowtie$ (PS $\bowtie$ S))}** // **two execution plans**
q_2**:((PS $\bowtie$ S) $\bowtie$ N)** // **one execution plan**

The remaining execution plans are just indicated in figure 2b by dashed lines.

In this paper, we use the AND-OR view graph to compactly represent alternative query plans and exhibit common subexpression. For more details about constructing the AND-OR view graph for the queries of workload, we may refer the reader to [28].

Our motivation to use the AND-OR representation rather than the AND representation since the latter makes local optimal choices, and may miss global optimal plans. The choice of materialized views must be done in conjunction with choosing execution plans for queries. For instance, a plan that seems quite inefficient could become the best plan if some intermediate result of the plan is chosen to be materialized and maintained as the following example demonstrates it.

Example. Let us consider the views P-PS-S and PS-S-N which are respectively computed by using the plan ((P $\bowtie$ PS) $\bowtie$ S) and the plan ((PS $\bowtie$ S) $\bowtie$ N), as it is shown in figure 2a. These execution plans represent the optimal plans for q_1 and q_2. However, if we choose the alternative plan (P $\bowtie$ (PS $\bowtie$ S)) to compute the view P-PS-S, the view PS-S becomes a common subexpression (see figure 2b). It can be computed once and used for both queries q_1 and q_2. This alternative with sharing of the view PS-S may be the global optimal choice.

In the context of view maintenance, common sub-expressions can be exploited to find an efficient plan for maintenance of a set of views. Indeed, the view P-PS-S may also be used for sharing updates and hence reducing the view maintenance cost.

Note that in the AND-OR view graph, each equivalence node, which represents a candidate view v_i to materialization, has the following parameters associated to it: Query cost Qc (the evaluation cost of the cheapest embedded expression AND-DAG for v_i), maintenance cost Mc (the cost required for updating v_i when the related base relations are changed), reading cost Rc, query frequency fq (if the equivalence node is a root node) and the update frequency fu (the frequency of updating v_i in response to change to the underlying data). To each operation node op_i, that represents a relational operator, a cost is associated with it which is the cost incurred during the computation of the parent node of op_i from the children nodes of op_i.

The view selection problem for AND-OR view graphs can be formulated as follows: *Given an AND-OR view graph G, a maximum storage space* Sp_{max} *(available space), a total view maintenance limit* U_{max} *(available maintenance time), the problem is to select a set of views to be materialized MV, a subset of the equivalence nodes of G, that minimizes the total query cost, under the constraint that the total space occupied by MV is less or equal than* Sp_{max} *and the total maintenance time of MV (i.e., view maintenance cost) is less or equal than* U_{max}.

5 Our View Selection Approach

Let us now introduce the constraint satisfaction model that we have proposed for the view selection problem. We then present the search strategy that we have defined within the constraint solver for optimization purpose.

5.1 Modeling View Selection Problem as a Constraint Satisfaction Problem (CSP)

This section describes how to model the view selection problem as a CSP. Then, its resolution is supported automatically by the constraint solver. In the following, we define all the symbols as well as the variables that we have used in our constraint satisfaction model.

- G. The AND-OR view graph described in the previous section.
- $Q(G)$. The views which correspond to the query results (the root nodes in the AND-OR view graph G).
- $V(G)$. The set of views in G which are candidate to materialization.
- $U(v_i)$. The set of updates on v_i in response to changes of the associated base relations.
- $\delta(v_i, u)$: The differential result of view v_i with respect to update u.
- $f_q(v_i)$. The access frequency or importance of the associated view (or query) v_i.

- $f_u(v_i)$. The frequency of propagating the changes of each associated base relation to the view v_i.
- Sp_{max}. The maximum storage space that can be used to view materialization.
- U_{max}. The time that can be allotted to keep up to date the materialized views.
- $size(v_i)$. The size of the view v_i in terms of number of bytes.

CSP Variables and Their Domains

- Mat_{v_i}. The materialization variable which denotes for each view v_i (equivalence node in the AND-OR view graph G), if it is materialized or not materialized. It is a binary variable, $d_{Mat_{v_i}} = \{0,1\}$ (0: v_i is not materialized, 1: v_i is materialized).
- $Qc(v_i)$. The query cost corresponding to the view v_i. The domain is a finite subset of $\mathbb{N}^*$ such as $d_{Qc(vi)} \subset \mathbb{N}^*$.
- $Mc(v_i)$. The maintenance cost corresponding to a view v_i, where $d_{Qc(vi)} \subset \mathbb{N}^*$.

The view selection problem can be formulated by the following constraint satisfaction model. It consists in specifying in a declarative way the CSP variables, their domains, and the constraints that are over them.

$$minimize \sum_{v_i \in Q(G)} \Big(f_q(v_i) * Qc(v_i) \Big) \tag{1}$$

$$subject\ to \sum_{v_i \in V(G)} \Big(Mat_{v_i} * size(v_i) \Big) \leq Sp_{max} \tag{2}$$

$$\sum_{v_i \in V(G)} \Big(Mat_{v_i} * f_u(v_i) * Mc(v_i) \Big) \leq U_{max} \tag{3}$$

In our approach, the main objective is the minimization of the total query cost. It is computed by summing over the cost of processing each input query rewritten over the materialized views. Constraints (2) and (3) state that the views are selected to be materialized under a limited amount of resources. Constraint (2) ensures that the total space occupied by the materialized views is less than or equal to the maximum storage space capacity. Constraint (3) guarantees that the total maintenance cost of the set of materialized views is less than or equal to the total view maintenance cost limit.

The query and maintenance costs corresponding to a view are implemented by using a depth-first traversal of the AND-OR view graph. We have been inspired by the formulas described in [24,28] to compute these two costs. Note that the query and maintenance costs corresponding to a base relation are equal to zero. They may be formulated as follows.

Query Cost

$$Qc(v_i) \quad = \begin{cases} CCost(v_i) \;\; if \;\; Mat_{v_i} = 0 \\ Rc(v_i) \; otherwise \end{cases} \tag{4}$$

$$where$$
$$CCost(v_i) = \min_{op_j \in child(v_i)} \left(cost(op_j) + \sum_{v_k \in child(op_j)} Qc(v_k) \right) \tag{5}$$

Constraint (4) states that the query cost corresponding to each given view in the AND-OR view graph is the minimum cost paths from the view to its related base relations or views. The reading cost is considered if the view has been materialized. Constraint (5) ensures that the minimum cost path is selected for computing a given view. Each minimum cost path includes all the cost of executing the operation nodes on the path and the query cost corresponding to the related bases relations or views.

View Maintenance Cost

$$Mc(v_i) \quad = \begin{cases} 0 \;\; if \;\; Mat_{v_i} = 0 \\ \sum_{u \in U(v_i)} Mcost(v_i, u) \; otherwise \end{cases} \tag{6}$$

$$where$$
$$Mcost(v_i, u) = \min_{op_j \in child(v_i)} \Bigg(cost(op_j, u) + \sum_{v_k \in child(op_j)} UCost(v_k, u) \Bigg) \tag{7}$$

$$UCost(v_k, u) = \begin{cases} Mcost(v_k, u) \;\; if \;\; Mat_{v_k} = 0 \\ \delta(v_k, u) \; otherwise \end{cases} \tag{8}$$

Constraint (6) guarantees that there is no maintenance cost if the view has not been materialized. Otherwise, the view maintenance cost is computed by summing the number of changes in the base relations from which the view is updated. We assume incremental maintenance to estimate the view maintenance cost. Therefore, the maintenance cost is the differential results of materialized views given the differential (updates) of the bases relations. Constraints (7) and (8) insure that the best plan with the minimum cost will be selected to maintain a view. The view maintenance cost is computed similarly to the query cost, but the cost of each minimum path is composed of all the cost of executing the operation nodes with respect to the updates on the path and the maintenance cost corresponding to the related base relations or views.

5.2 Search Strategy

A key ingredient of any constraint satisfaction approach is an efficient search strategy. As mentioned in Section 3.2, the search is organized as an enumeration tree, where each node corresponds to a subspace of the search. The tree is progressively constructed by applying series of branching strategies that define the way to branch from a tree search node. In the constraint solver, branching has been applied to decision variables. In our constraint satisfaction model, the materialization variable Mat_{v_i} is the decision variable since the aim of the view selection problem is to decide which views to materialize. The most common branching strategies in the constraint solver are based on the assignment of a selected variable to one or several selected values (one assignment in each branch). Variable selector defines the way to choose a non instantiated variable on which the next decision will be made. Once the variable has been chosen, the solver has to compute its value.

5.2.1 The Default Search Strategy. The default search strategy is applied to the decision variables of the solver when no search strategy is specified. The default strategy selects the decision variables to be instantiated by using the following branching strategies.

Variable selection heuristic: DomOverWDeg. The strategy selects the variable Mat_{v_i} with the smallest ratio r:

$$r = \frac{dom}{w * deg}$$

where dom is the current domain size, deg is the current number of non instantiated constraints involving the variable, and w the sum of the counters of the failures caused by each constraint from the beginning of the search. To each variable $Mat(v_i)$ are associated, at any time the dom, deg and w values.

Value selection heuristic: MinVal. The variable Mat_{v_i} which has been chosen (by applying the variable selection heuristic) is then assigned, in the first branch, to its smallest value:

$$val = \min(d_{Mat_{v_i}})$$

In the next branch, the value val is removed from the variable domain $d_{Mat_{v_i}}$.

5.2.2 Our Own Search Strategy. As mentioned in Section 3.2, constraint programming offers facilities to control the search behavior. Defining our own search strategy is very important since a well-suited search strategy can reduce the number of expanded nodes and hence the time that the solver takes to find solutions to the view selection problem. In the following we describe the variable and value selection heuristics that we have defined in the search strategy.

Variable and value selection heuristics. Our aim is to minimize the query cost with a constraint on update time (maintenance cost constraint) and storage

space (space constraint). Low query cost can be obtained by materializing all the queries of the workload (materializing the root level in the AND-OR view graph). In this case the view maintenance cost will be high. Low view maintenance cost can be achieved by leaving all the views virtual and in this case the query cost will be high (replicating the base relations which are in the leaf level of the AND-OR view graph). For this matter, our strategy consists in finding an intermediary level for each query tree in the AND-OR view graph that optimizes the query cost without violating the maintenance cost and space constraints. Therefore, our strategy is based on the notion of level in the AND-OR view graph [4]. For this purpose, each view (equivalence node) is associated to a level, which is defined as follows:

$$level(baserelation) = 0$$
$$level(view) = \max_{v_c \in child(view)} level(v_c) + 1$$

As presented in the code below, we explain how to compute for each query the relative query cost reduction associated to the different levels in the query tree.

```
levels = ∅ //set of levels with their cost saving
for each q in Q(G) do
  levelCS = ∅//Map : key = level; val = cost saving
  // each view in the query tree is associated to a level
  for each l in AllLevels(q) do
    space = 0
    maint = 0
    for each v in AllViews(l)  do
      space = space + size(v)
      maint = maint + Mc(v)
    end for
    if space ≤ Sp_Max and maint ≤ U_Max then
      LevelCostSaving(q, l)
      //LevelCostSaving is defined as the relative
      //query cost reduction when the views associated
      // to level l are materialized
    else
      LevelCostSaving(q, l) = −1
    end if
    levelCS.put(l, LevelCostSaving)
  end for
  levels = levels ∪ {levelCS}
end for
```

In order to guide the search to the optimal solution, the variable selector has to start by instantiating the materialization variables of the recommended

views. These views are those associated to the levels that minimize the query cost subject to space and maintenance cost constraints. For this purpose, we sort the query levels according to their *LevelCostSaving* in descending order (as it is presented below). We iterate over the sorted set starting with the levels which have the highest query cost reduction. We then store each view associated to these levels in the variable MV.

```
//sort the levels according to their LevelCostSaving in
//descending order
LSort = SortLevels(levels)
for each l_s in LSort  do
  for each v_s in l_s  do
    MV = MV ∪ {Mat_{v_s}}
  end for
end for
```

Finally, the variable selector will choose the materialization variables to be instantiated in the order they appear in MV. Once the variable has been chosen, the value selector will assign the materialization variable to its highest value: $max(d_{Mat_{v_i}})$. Note that these variable and value heuristics do not inhibit the solver to compute solutions in which it will start by materializing another set of views. By defining these heuristics in the search strategy, we expect the solver to converge faster to the optimal solution and avoid browsing a large number of inferior solutions.

6 Performance Evaluation

In this section, we evaluate the performance of our approach through experimentations over the database schema of the TPC-H benchmark [2]. Our approach takes as input a set of selection-projection-join (SPJ) queries that may involve aggregation and group by clause as well. For each query, we consider all possible execution plans which represent its execution strategies. Then, all the queries are merged into the same graph (see Section 4) in order to detect the overlapping and capture the dependencies among them. Our approach produces as output the set of materialized views. The performance of our approach was evaluated by measuring the gain in solution quality obtained by the materialized views.

The rest of this section is organized as follows. In Section 6.1, we describe our experimental setup, and the randomized method used for comparison. In Section 6.2, we study the impact of variable and value selection heuristics on the search space explored by our approach. In Section 6.3, we first report experimental results when the view selection is decided under resource constraints and we present the results on performance by increasing the number of queries. Then, we evaluate the effect of the frequency of queries and updates as well as the query complexity on performance. In Section 6.4, we study the benefit of using

materialized views to improve query performance. Finally, we summarize the performance results in Section 6.5.

6.1 Experimental Setup

We have implemented our approach and compared it with a randomized method i.e., genetic algorithm . The latter was chosen for comparison since it has been argued that the genetic algorithm provides a good balance between the computing costs that an algorithm incurs for finding the materialized views and the gain to be realized in query processing by materializing these views (see Section 2). All the algorithms are implemented in Java and all the experiments were carried out on an Intel Core 2 Duo P8600 CPU @ 2.40 GHz machine running with 3GB of RAM and Windows XP Professional SP3.

In order to solve the view selection problem as a constraint satisfaction problem, we have used the latest powerful version of CHOCO [1] (knowing that the constraint solvers are structured around annual competitions [17]). For the genetic algorithm, we have implemented the one presented in [5] by incorporating space and maintenance cost constraints into the algorithm and without taking into account the data placement. In order to let the genetic algorithm converge quickly, we generated an initial population which represents a favorable view configuration rather than a random sampling. Favorable view configuration such as the views which minimize the query cost without violating space and maintenance cost constraints are most likely selected for materialization.

To evaluate the performance of view selection methods, we measure the following metric.

1. **Solution Quality.** The performance of view selection methods was evaluated by measuring the solution quality which results from evaluating the quality of the obtained set of materialized views in terms of cost saving. In the experimental results, the solution quality denoted by Q_s is computed as follows:

$$Q_s = \frac{WM - \sum_{v_i \in Q(G)} \big(f_q(v_i) * Qc(v_i)\big)}{WM - ALLM} \tag{9}$$

   Where WM is the total query cost obtained using the "WithoutMat" approach which does not materialize views and always recomputes queries, $AllM$ is the "AllMat" approach which materializes the result of each query of the workload. The "WithoutMat" and "AllMat" approaches are used as a benchmark for our normalized results. As defined in Section 5, $Qc(v_i)$ is the query cost corresponding to the view v_i and $f_q(v_i)$ is the frequency of the view v_i.

2. **Space constraint.** In the case where the view selection problem is decided under a space constraint, the total space occupied by the materialized views has to be less than or equal to the maximum storage space Sp_{max} . Similar

to [10] Sp_{max} is computed as a function of the size of the associated query workload.

$$Sp_{max} = \alpha * Sp_{AllM} \tag{10}$$

where Sp_{AllM} is the size of the whole workload and α is a constant. In our experiments, we assume the case where the view selection is studied under restrictive constraints and hence we set α to 10%. We also examine the case where the constraints are not very tight and at that case α was set to 30%.

3. **Maintenance Cost Constraint.** In the maintenance cost constrained model, the total maintenance cost of the set of materialized views has to be less than or equal to the total view maintenance cost limit U_{max}. As in previous work [10], U_{max} is calculated as a function of the total maintenance cost when all the queries are materialized.

$$U_{max} = \beta * Mc_{AllM} \tag{11}$$

where Mc_{AllM} is the total maintenance cost when the result of each query of the workload is materialized and β is a constant. The value of β was set similar to α (see above).

4. **Runtime**. The Runtime which we consider here is the time that MySQL server takes to compute query results using materialized view. This metric has been used in Section 6.4 to measure the running time of the query workload given a set of materialized views. Thus, the runtime is a good metric to study the benefit that materialized views found by our approach bring to query evaluation. It is also a good indicator for comparing the performance of our approach to those of the genetic algorithm.

6.2 Impact of Variable and Value Selection Heuristics

Here, we study the impact of variable and value selection heuristics that we have presented in Section 5.2.2, on the search space explored by our approach. To evaluate this, we attempted to compare the solution quality found by the constraint solver in the case where (i) the default search strategy is used and (ii) the variable and value selection heuristics that we have defined in Section 5.2.2 are implemented in the search strategy. As mentioned in Section 3.2, the constraint solver (CHOCO Solver) can find a set of feasible solutions in which all the constraints are satisfied before reaching the optimal solution. In this case, we use *timeout* condition to evaluate the quality of the different solutions found by the solver. A workload of 20 queries suffices to illustrate this. α and β, which define respectively the storage space and the view maintenance cost limits, was set to 30%. The results are shown in figure 3. The solver is left to run until reaching the optimal solution. *default search* denotes the default search strategy while *custom search* requires the variable and value selection heuristics that we have defined in the search strategy. We can observe from figure 3 that the time

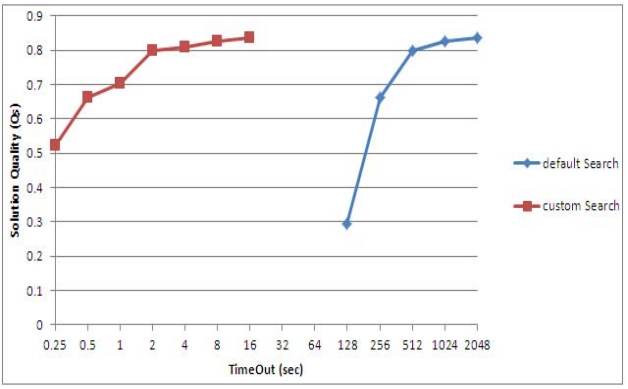
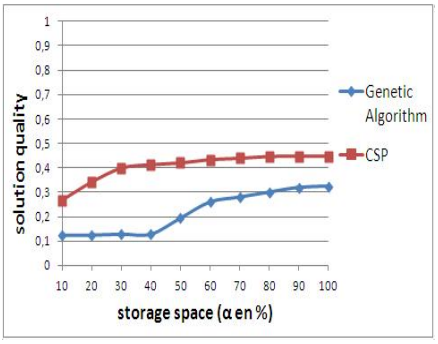
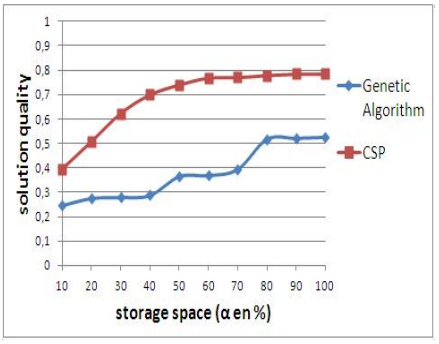


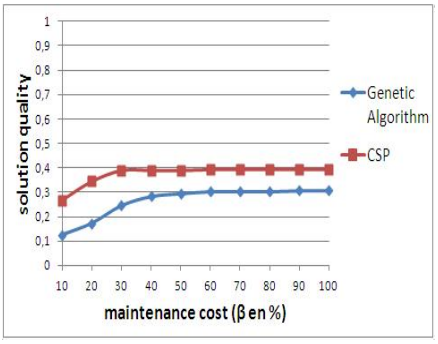
Fig. 3. Impact of heuristics on the search



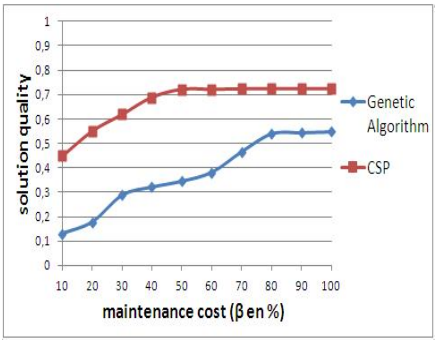
(a) β=10%



(b) β=30%



(c) α=10%



(d) α=30%

Fig. 4. Solution quality while varying the space or the maintenance cost constraint

that a solver incurs in the presence of *custom search* for finding near optimal and optimal solutions is significantly reduced. This is because the variable and value selection heuristics that we have defined in the search strategy reduce significantly the search space explored by the CHOCO solver. Consequently, our approach can provide high solution quality in a short time. In the following experiments, we use the *custom search* in the constraint satisfaction model.

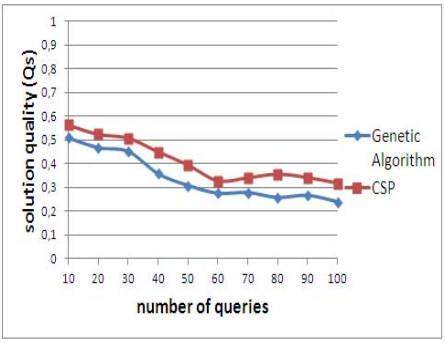
6.3 Solution Quality: Our Approach versus Genetic Algorithm

In this section, we examined the effectiveness of our approach by measuring the gain in solution quality obtained by using our approach versus the genetic algorithm. First, we compare the performance of our approach to those of the genetic algorithm for various values of storage space and maintenance cost limits and then we present the impacts on performance by increasing the number of queries. We also evaluate the solution quality found by view selection methods with respect to different query and update distributions. Finally, we evaluate our approach and the genetic algorithm according to query complexity. In order to allow a fair comparison with the genetic algorithm and since our approach is able to provide a solution at any time, the CHOCO solver was left to run until the convergence of the genetic algorithm in the following experiments. More precisely, the *timeout* condition was set to the time required by the genetic algorithm to solve the view selection problem. Since the heuristic based search strategy allows the solver to find a high solution quality very fast (as described in the previous section) and the genetic algorithm requires an amount of time to converge, we expect to achieve significant performance gains in comparison with the genetic algorithm in terms of cost saving.

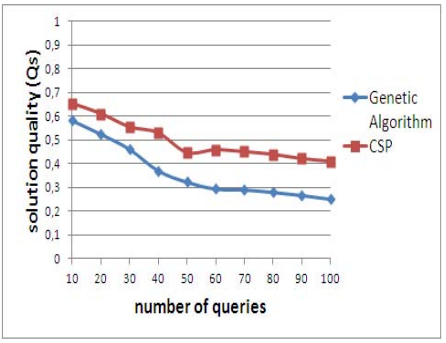
6.3.1 Resource Constraints. In this experiment, we first examine the impact of space and maintenance cost constraints on solution quality. For this evaluation, we consider a workload of 50 queries. Recall that for each query, we consider all possible execution plans which represent its execution strategies. The query and update frequencies are at scale 1. The values of α and β which define respectively the storage space capacity and the view maintenance cost limit are varied from 10% to 100%. All the results are shown in figure 4.

Figure 4a and Figure 4b investigate respectively the influence of space constraint on solution quality for each value of α where β was set to 10% and 30%, while figure 4c and figure 4d examine respectively the impact of maintenance cost constraint on solution quality for each value of β where α was set to 10% and 30%. We note from these experiments that the quality of the solutions produced by our approach and genetic algorithm improves when α (see figure 4a and figure 4b) or β (see figure 4c and figure 4d) increases. However, there is no improvement in the solution quality from certain values of α or β because the maintenance cost constraint or the space constraint becomes the significant factor.

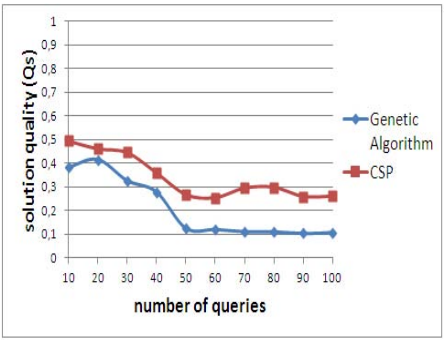
We also observe from figure 4 that our approach provides better solution quality in the case where the view selection is decided under a maintenance cost constraint (i.e., $Q_s \approx 0.8$ when α=100% and β=30% in figure 4b while $Q_s \approx 0.7$ when β=100% and α=30% in figure 4d). The reason is the maintenance cost of a view may decrease with selection of other views for materialization. Hence, there



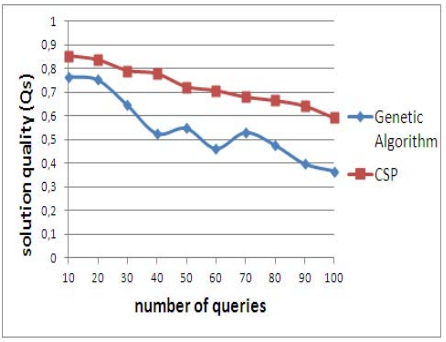
(a) α=10%



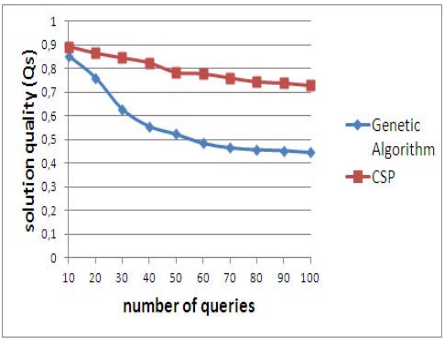
(b) β=10%



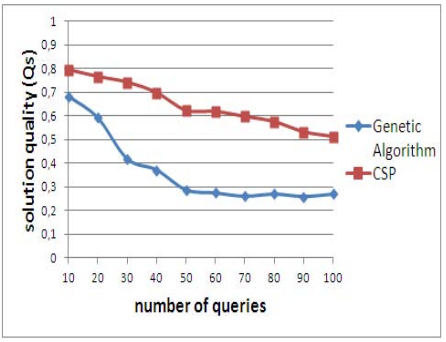
(c) α=10% and β=10%



(d) α=30%



(e) β=30%



(f) α=30% and β=30%

Fig. 5. Solution quality on large workloads under different resource constraints

is time to update more views. This non monotonic nature of view maintenance cost is formally defined in [8].

Finally, we conclude from these experiments that our approach outperforms the genetic algorithm for different values of α and β in terms of cost saving. Indeed, we can see that our approach generates solutions with cost saving up to 2 times more than the genetic algorithm.

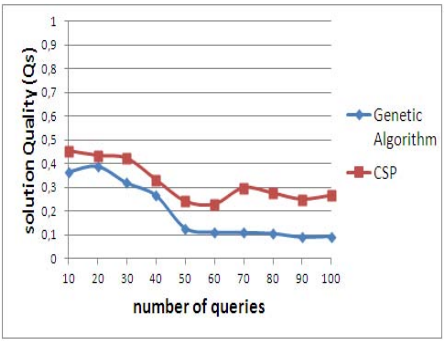
6.3.2 Large Query Workload. Let us now evaluate the performance of our approach and the one of genetic algorithm on larger query workload. For this purpose, we generated workloads of 10, 20, 30, 40, 50, 60, 70, 80, 90 and 100 queries. The solution quality of our approach and the genetic algorithm is evaluated when the view selection is decided under the case where (i) only the space constraint is considered (see figure 5a and figure 5d); (ii) the limiting factor is the view maintenance cost(see figure 5b and figure 5e); and (iii) both maintenance cost and space constraints exists (see figure 5c and figure 5f). On each of these cases, we consider the case where the resource constraints become very tight (α and/or $\beta = 10\%$) as well as the case where we relax them (α and/or $\beta = 30\%$).

For this collection of experiments, we make the following observations. Our approach provides in all the cases better performances in terms of the solution quality while varying the number of queries. For example for a workload of 100 queries where α and β was set to 30% (see figure 5f), our approach provides a cost saving of 24% more than the genetic algorithm ($Qs_{CSP} = 0.512$ while $Qs_{GeneticAlgorithm} = 0.27$). Another remark based on figure 5 is that in our approach the gain in solution quality tends to be relatively more significant when

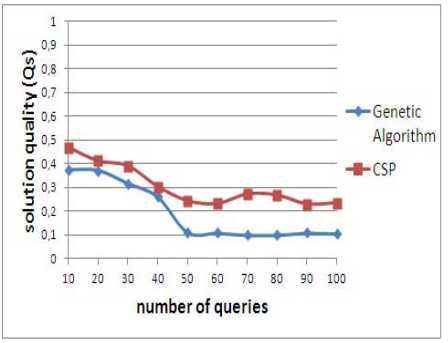
Table 1. Distribution of query and update frequencies

q_{random}	The values of the query frequencies have been assigned randomly to each query of the workload.
$q_{uniform}$	All the queries of the workload have the same query frequency
$q_{gaussian}$	Queries from certain levels have higher probability to be queried. The frequency distribution is normal with $\mu = 1/2$ and $\sigma = 1$.
u_{random}	The values of the update frequencies have been assigned using a random distribution.
$u_{uniform}$	All the views in the AND-OR view graph have the same update frequency
$u_{gaussian}$	The views which are at the lower level of the AND-OR view graph have higher probability to be updated than those which are on the upper level (guassian distribution with $\mu = 1/2$ and $\sigma = 1$).

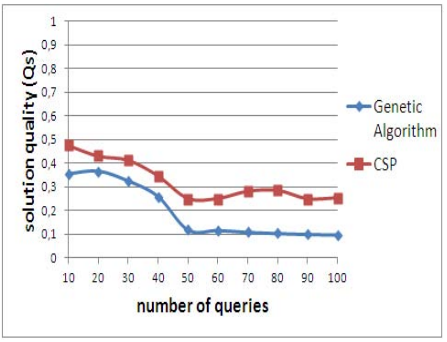
we have more resource constraints. For instance, the gain in solution quality obtained by our approach is up to 10% (in figure 5a) and 16% (in figure 5b) more than the genetic algorithm. While this gain is up to 18% in figure 5c. This is because the idea of constraint programming is to solve problems by stating constraints and the search space is reduced when there are more constraints.



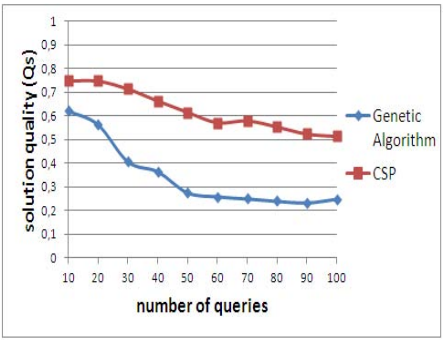
(a) q_{random}, α=10% and β=10%



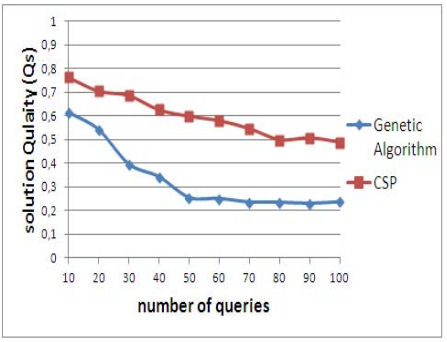
(b) $q_{uniform}$, α=10% and β=10%



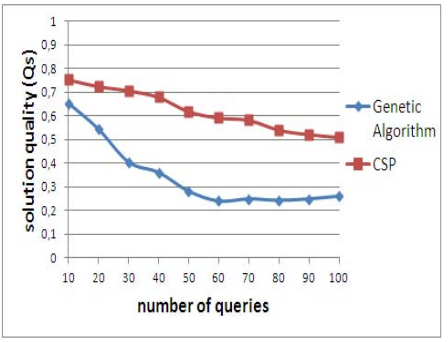
(c) $q_{gaussian}$, α=10% and β=10%



(d) q_{random}, α=30% and β=30%



(e) $q_{uniform}$, α=30% and β=30%



(f) $q_{gaussian}$, α=30% and β=30%

Fig. 6. Solution quality for different query distributions

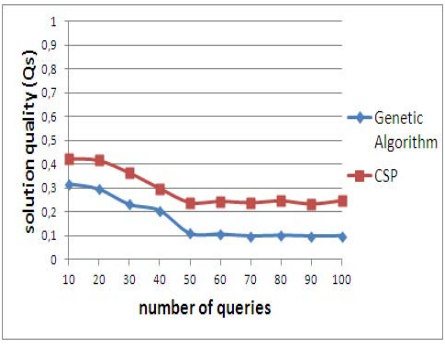
This result is similar to the case where we relax the constraints (see figures 5d, 5e and 5f).

6.3.3 Query and Update Distributions. We now study the behavior of view selection methods while varying the query and update frequencies. For this purpose, we generated different query and update distribution to simulate various workloads (see table 1). The random distribution assigns random values to query or update frequencies. While, the uniform distribution simulates cases where all views (or queries) have equal probability to be queried and updated. The last distribution which is the gaussian distribution favors views (or queries) from lower levels in the AND-OR view graph that have higher probability to be queried or updated. For example, queries of the TPC-H benchmark which contain less relational operators have higher probability to be queried.

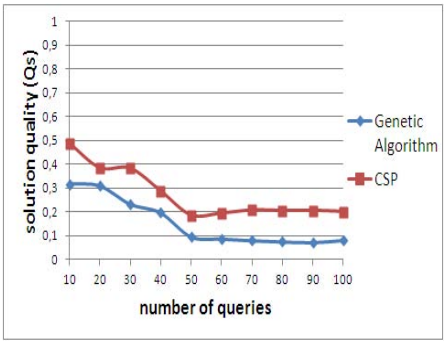
Figures 6 illustrates the quality of the solutions produced by the two methods for different query distributions (q_{random}, $q_{uniform}$, $q_{gaussian}$). In the first combination, α and β were set to 10% (see figures 6a, 6b and 6c). While, for the other combination, α and β were set to 30% (see figures 6d, 6e and 6f). The update frequencies are at scale 1. We have made the same experiments for different update distributions in which the query frequencies was at scale 1 (see figure 7).

We can see that the quality of the solutions found by our approach is always better than those of the genetic algorithm for different query and update distributions. For example, in figure 6 and in the worst case which arises at the random workload (q_{random};α=10%;β=10%), our approach provides solutions with a cost saving of 4% more than the genetic algorithm. While, in the best case which arises at the gaussian workload ($q_{gaussian}$;α=30%;β=30%), the cost saving is 35% more than the genetic algorithm.

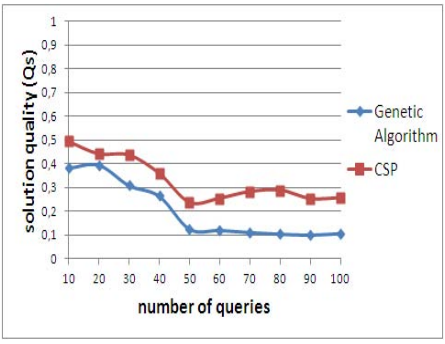
6.3.4 Query Complexity. We study the effect of query complexity on view selection performance. More specifically, we evolved the number of join operators $N_{Join_{Op}}$ for each query of the workload since the complexity of binary operators is more important than the one of unary operators. This results to three different workloads: (i) c_query_01 ($N_{Join_{Op}} < 2$); (ii) c_query_02, ($2 \leq N_{Join_{Op}} < 4$); and (iii) c_query_03, ($N_{Join_{Op}} \geq 4$). We run experiments with a workload of 50 queries and we measure the gain in solution quality according to the set of the obtained materialized views. The frequencies for access and update are at scale 1. Figure 8 shows the cost saving found by our approach and the genetic algorithm for both cases: (i) α and β was set to 10% (see figure 8a) and (ii) α and β was set to 30% (see figure 8b). We can see that our approach produce the best results. Indeed, our approach provides a cost saving up to 27.2% when α and β was set to 10% and 63.3% when α and β was set to 30%. While the genetic algorithm achieve a cost saving of only 12.9% when α and β was set to 10% and 29.3% when α and β was set to 30%. We also observe, in the graphic depicted in figure 8, that the quality of the solutions produced by our approach



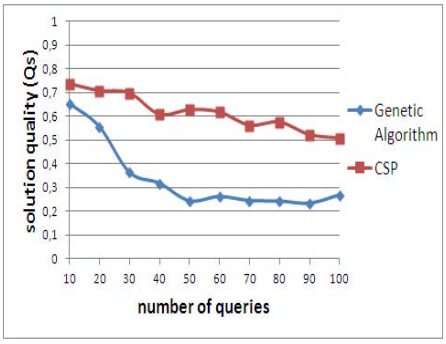
(a) u_{random}, α=10% and β=10%



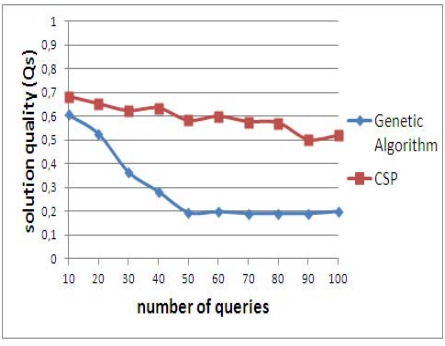
(b) $u_{uniform}$, α=10% and β=10%



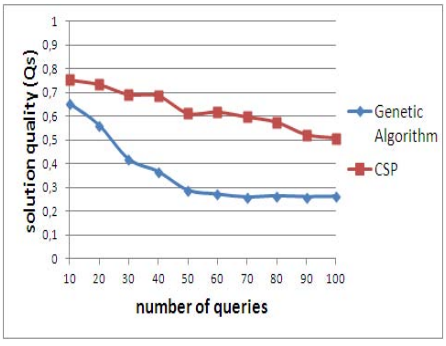
(c) $u_{gaussian}$, α=10% and β=10%



(d) u_{random}, α=30% and β=30%



(e) $u_{uniform}$, α=30% and β=30%



(f) $u_{gaussian}$, α=30% and β=30%

Fig. 7. Solution quality for different update distributions

slightly decrease with an increasing complexity of the query workload. Hence, we confirm that the performance of our approach is not significantly influenced by an increasing of query complexity.

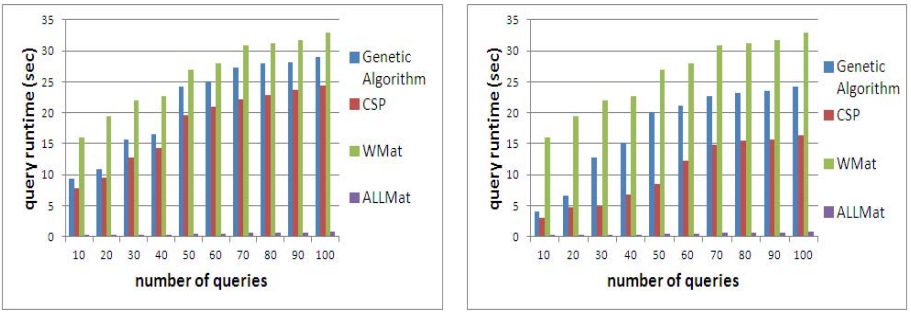
(a) α=10% and β=10% (b) α=30% and β=30%

Fig. 8. Query complexity on view selection performance

6.4 Query Performance Using Materialized Views

In this section, we study the benefit of using materialized views to improve query performance. For a workload involving 10, 20, 30, 40, 50, 60, 70, 80, 90 and 100 queries, we materialized the views proposed by our approach and the genetic algorithm. Then, we run the query workload using these views. We also consider the two basic strategies that we have defined above: the "WithoutMat" and the "AllMat" approaches. Recall that the "WithoutMat" approach does not materialize views and always recomputes queries. While the "AllMat" approach materializes the result of each query without any resource constraint. The frequencies for access and update are at scale 1. In order to measure the query runtime, the experiments were performed on MySQL server through JDBC interface. The query runtime is expressed in seconds (sec).

The results are shown in figure 9. The view selection has been decided under space and maintenance cost constraints: (i) α and β was set to 10% in figure 9a and (ii) α and β was set to 30% in figure 9b. The results indicate that the benefit of using materialized views is significant. Indeed, queries using our proposed views or those of the genetic algorithm are evaluated faster in comparison with



(a) α=10% and β=10% (b) α=30% and β=30%

Fig. 9. Query runtime using materialized views

the "WithoutMat" approach. We can also see that our approach provides the better quality of the obtained set of materialized views. For instance as can be seen in figure 9b, when comparing the runtimes of the workload of 100 queries, our approach requires $\approx 16seconds$ while genetic algorithm takes $\approx 24seconds$. According to the equation (9) in section 6.1,

$$Qs_{CSP} = 0.518 = \frac{32.86(WM) - 16.297(CSP)}{32.86(WM) - 0.892(ALLM)} \quad (12)$$

$$Qs_{GeneticAlgorithm} = 0.273 = \frac{32.86(WM) - 24.126(GeneticAlgorithm)}{32.86(WM) - 0.892(ALLM)} \quad (13)$$

This result confirms our expectation in section 6.3.2 that for a workload of 100 queries where α and β was set to 30%, our approach provides a cost saving of 24% more than the genetic algorithm ($Qs_{CSP} = 0.512$ while $Qs_{GeneticAlgorithm} = 0.27$). Another important remark is that our approach is able to provide materialized views that produce higher cost savings even if the underlying cost model is simplified (see section 3). Thus, our approach is robust toward simplified cost models which is an important requirement for a practical solution to the view selection problem.

6.5 Concluding Remarks

Our experiments show that our approach outperforms the genetic algorithm in many cases. We achieve impressive cost saving factors when (i) we study the view selection under resource constraints, (ii) we increase the number of queries and (iii) we simulate various query workloads. We also show the efficiency of our approach when we run the query workloads on MySQL server i.e., queries using our proposed views are evaluated faster in comparison with those found by the genetic algorithm. The experiment results confirm our expectation that our own search strategy allows our approach to achieve significant performance gains in comparison with the genetic algorithm.

7 Conclusion

The most efficient algorithm proposed so far for deciding which views to materialize is the genetic algorithm that provides the best trade-off between quality of solutions and execution time. However, there is no guarantee of performance because the probabilistic behavior of the genetic algorithms does not insure to find the global optimum. Besides, the quality of the solution depends on the set-up of the algorithm as well as the extremely difficult fine-tuning of algorithm that must be performed during many test runs.

In this paper, we proposed a declarative approach which simply modeled the view selection problem as a CSP without the need of being interested in the way the problem is solved. Indeed, its resolution was supported automatically by the constraint solver. We also designed a heuristic search strategy within the constraint solver to reduce the solution space and hence the execution time. The experiment results confirm our expectation that our own search strategy allows

our approach to achieve significant performance gains in comparison with the genetic algorithm.

More recently, the view selection has been investigated in data placement in a distributed setting [5]. For this purpose, we have extended our constraint satisfaction model to deal with the distributed setting. The formulation of the view selection problem in a distributed context can be found in our recent work [22]. As a future work, we are planning to solve the view selection problem in a large scale distributed environments such as peer to peer or cloud computing environments.

In our proposals, all queries are assumed to be known and given in advance and there is a frequency of occurrence associated with each query. One line of recent research [12] has explored the problem of identifying subject area specific queries from which frequent queries are selected. As a result, they obtain significant performance improvements when processing queries. However, the proposed approach is based on a given workload and chooses accordingly the set of views to materialize. In order to respond to the changes in the query workload over time, views need to be selected continuously. Consequently, the dynamic view selection issue will be a part of our planned future work while studying the view selection in large scale distributed environments.

References

1. Choco, open-source software for constraint satisfaction problems, `http://www.emn.fr/z-info/choco-solver`
2. The TPC benchmark H (TPC-H), `http://www.tpc.org/tpch/spec/tpch2.14.3.pdf`
3. Agrawal, S., Chaudhuri, S., Narasayya, V.R.: Automated selection of materialized views and indexes in sql databases. In: VLDB, Cairo, Egypt, pp. 496–505 (2000)
4. Baril, X., Bellahsene, Z.: Selection of materialized views: A cost-based approach. In: Eder, J., Missikoff, M. (eds.) CAiSE 2003. LNCS, vol. 2681, pp. 665–680. Springer, Heidelberg (2003)
5. Chaves, L.W.F., Buchmann, E., Hueske, F., Böhm, K.: Towards materialized view selection for distributed databases. In: Proceedings of the 12th International Conference on Extending Database Technology: Advances in Database Technology, EDBT 2009, pp. 1088–1099. ACM, New York (2009)
6. Du, W., Krishnamurthy, R., Shan, M.C.: Query optimization in heterogeneous dbms. In: Proc. of VLDB. Vancouver, British Columbia, Canada, pp. 277–291 (1992)
7. Gupta, H.: Selection of views to materialize in a data warehouse. In: ICDT, Delphi, Greece, pp. 98–112 (1997)
8. Gupta, H., Mumick, I.S.: Selection of views to materialize under a maintenance cost constraint. In: ICDT, Jerusalem, Israel, pp. 453–470 (1999)
9. Harinarayan, V., Rajaraman, A., Ullman, J.D.: Implementing data cubes efficiently. In: SIGMOD Conference, Montreal, Canada, pp. 205–216 (1996)
10. Kalnis, P., Mamoulis, N., Papadias, D.: View selection using randomized search. Data Knowl. Eng. 42(1), 89–111 (2002)
11. Karloff, H.J., Mihail, M.: On the complexity of the view-selection problem. In: PODS, Philadelphia, Pennsylvania, USA, pp. 167–173 (1999)

12. Vijay Kumar, T.V., Dubey, G., Singh, A.: Frequent queries selection for view materialization. In: Meghanathan, N., Nagamalai, D., Chaki, N. (eds.) Advances in Computing & Inform. Technology. AISC, vol. 177, pp. 521–530. Springer, Heidelberg (2012)
13. Vijay Kumar, T.V., Kumar, S.: Materialized view selection using genetic algorithm. In: IC3, pp. 225–237 (2012)
14. Vijay Kumar, T.V., Kumar, S.: Materialized view selection using iterative improvement. In: Meghanathan, N., Nagamalai, D., Chaki, N. (eds.) Advances in Computing & Inf. Technology. AISC, vol. 178, pp. 205–213. Springer, Heidelberg (2012)
15. Kumar, V.: Algorithms for constraint-satisfaction problems: A survey. AI Magazine 13(1), 32–44 (1992)
16. Labio, W., Quass, D., Adelberg, B.: Physical database design for data warehouses. In: Proceedings of the Thirteenth International Conference on Data Engineering, ICDE 1997, pp. 277–288. IEEE Computer Society, Washington, DC (1997)
17. Lecoutre, C., Roussel, O., van Dongen, M.R.C.: Promoting robust black-box solvers through competitions. Constraints 15(3), 317–326 (2010)
18. Lee, M., Hammer, J.: Speeding up materialized view selection in data warehouses using a randomized algorithm. Int. J. Cooperative Inf. Syst. 10(3), 327–353 (2001)
19. Ligoudistianos, S., Theodoratos, D., Sellis, T.K.: Experimental evaluation of data warehouse configuration algorithms. In: DEXA Workshop, Vienna, Austria, pp. 218–223 (1998)
20. Mackert, L.F., Lohman, G.M.: R* optimizer validation and performance evaluation for local queries. In: Proceedings of the 1986 ACM SIGMOD International Conference on Management of Data, SIGMOD 1986, pp. 84–95. ACM, New York (1986)
21. Mami, I., Bellahsene, Z.: A survey of view selection methods. SIGMOD Record 41(1), 20–29 (2012)
22. Mami, I., Bellahsene, Z., Coletta, R.: View selection under multiple resource constraints in a distributed context. In: Liddle, S.W., Schewe, K.-D., Tjoa, A.M., Zhou, X. (eds.) DEXA 2012, Part II. LNCS, vol. 7447, pp. 281–296. Springer, Heidelberg (2012)
23. Mami, I., Coletta, R., Bellahsene, Z.: Modeling view selection as a constraint satisfaction problem. In: Hameurlain, A., Liddle, S.W., Schewe, K.-D., Zhou, X. (eds.) DEXA 2011, Part II. LNCS, vol. 6861, pp. 396–410. Springer, Heidelberg (2011)
24. Mistry, H., Roy, P., Sudarshan, S., Ramamritham, K.: Materialized view selection and maintenance using multi-query optimization. In: SIGMOD Conference, Santa Barbara, California, USA, pp. 307–318 (2001)
25. De Raedt, L., Guns, T., Nijssen, S.: Constraint programming for itemset mining. In: KDD, Las Vegas, USA, pp. 204–212 (2008)
26. Rossi, F., van Beek, P., Walsh, T.: Handbook of Constraint Programming (Foundations of Artificial Intelligence). Elsevier Science Inc., New York (2006)
27. Roussopoulos, N.: The logical access path schema of a database. IEEE Trans. Software Eng. 8(6), 563–573 (1982)
28. Roy, P., Seshadri, S., Sudarshan, S., Bhobe, S.: Efficient and extensible algorithms for multi query optimization. In: SIGMOD Conference, Dallas, Texas, USA, pp. 249–260 (2000)
29. Sohn, J.-S., Yang, J.-H., Chung, I.-J.: Improved view selection algorithm in data warehouse. In: Kim, K.J., Chung, K.-Y. (eds.) IT Convergence and Security 2012. LNEE, vol. 215, pp. 921–928. Springer, Heidelberg (2012)
30. Theodoratos, D., Ligoudistianos, S., Sellis, T.K.: View selection for designing the global data warehouse. Data Knowl. Eng. 39(3), 219–240 (2001)

31. Theodoratos, D., Sellis, T.K.: Data warehouse configuration. In: VLDB, Athens, Greece, pp. 126–135 (1997)
32. Yang, J., Karlapalem, K., Li, Q.: Algorithms for materialized view design in data warehousing environment. In: VLDB, Athens, Greece, pp. 136–145 (1997)
33. Zhang, C., Yang, J.: Genetic algorithm for materialized view selection in data warehouse environments. In: Mohania, M., Tjoa, A.M. (eds.) DaWaK 1999. LNCS, vol. 1676, pp. 116–125. Springer, Heidelberg (1999)
34. Zhang, C., Yao, X., Yang, J.: An evolutionary approach to materialized views selection in a data warehouse environment. IEEE Transactions on Systems, Man, and Cybernetics, Part C 31(3), 282–294 (2001)

A Framework for Modeling, Computing and Presenting Time-Aware Recommendations

Kostas Stefanidis[1,3], Eirini Ntoutsi[2], Mihalis Petropoulos[2], Kjetil Nørvåg[3], and Hans-Peter Kriegel[2]

[1] Institute of Computer Science, FORTH, Heraklion, Greece
kstef@ics.forth.gr
[2] Institute for Informatics, Ludwig Maximilian University, Munich, Germany
{ntoutsi,petropoulos,kriegel}@dbs.ifi.lmu.de
[3] Department of Computer and Information Science, Norwegian University of Science and Technology, Trondheim, Norway
kjetil.norvag@idi.ntnu.no

Abstract. Lately, recommendation systems have received significant attention. Most existing approaches though, recommend items of potential interest to users by completely ignoring the temporal aspects of ratings. In this paper, we argue that time-aware recommendations need to be pushed in the foreground. We introduce an extensive model for time-aware recommendations from two perspectives. From a *fresh-based* perspective, we propose using different aging schemes for decreasing the effect of historical ratings and increasing the influence of fresh and novel ratings. From a *context-based* perspective, we focus on providing different suggestions under different temporal specifications. To facilitate user browsing, we propose an effective presentation layer for time-aware recommendations based on user preferences and summaries for the suggested items. Our experiments with real movies ratings show that time plays an important role in the recommendation process.

1 Introduction

Recommendation systems provide users with suggestions about products, movies, videos and a variety of other items. A popular category of recommendation systems is the collaborative filtering approaches (e.g., [21,11]) that try to predict the utility of items for a particular user based on the items previously rated by similar users. That is, users similar to a target user are first identified, and then, items are recommended based on the ratings of these users. Users are considered as similar if they buy common items as in case of Amazon or if they provide similar movie evaluations as in case of MovieLens.

Although there is a substantial amount of research in the area of recommendation systems [34], most of the approaches produce recommendations by ignoring the temporal information that is inherent in the ratings, since ratings are given at a specific point in time. Due to the fact that a huge amount of user preferences data is accumulated over time, it is reasonable to exploit the temporal information associated with these data in order to obtain more accurate and up to date

A. Hameurlain et al. (Eds.): TLDKS X, LNCS 8220, pp. 146–172, 2013.

recommendations. Our goal is to use the time information of the user ratings towards improving the predictions in collaborative recommendation systems. We consider two different types of time effects based upon the recency/freshness and the temporal context of the ratings and consequently, we propose two different time-aware recommendation models, namely the fresh-based and the context-based recommendations model.

The *fresh-based recommendations model* assumes that the most recent user ratings better reflect his/her current trends and thus, they should contribute more in the computation of the recommendations. To account for the recency of the ratings we distinguish between the *damped window model* that gradually decreases the importance of ratings over time and the *sliding window model* that counts only for the most recent ratings and ignores any previous historical information. As an example, consider a movie recommendation system that gives higher priority to new releases compared to other old seasoned movies (damped window model) or focuses solely on new releases (sliding window model).

From a different perspective, the *context-based recommendations model* offers different suggestions under different time specifications. The main motivation here, is that although user preferences may change over time they display temporal repetition, i.e., recur over time. As an example consider a tourist guide system that provides different suggestions for winter (typically ski resorts) and summer (typically sea resorts). Or, a restaurant recommendation system that might distinguish between weekdays (typically business lunches) and weekends (typically family lunches).

It is the purpose of this paper to provide a framework for time-aware recommendations that handles the different temporal aspects of recommendations through the fresh-based or the context-based model. Apart from the top-k recommendations extraction, we also focus on their effective presentation to the end user by adding structure in the results. Our goal is to minimize the browsing effort of the user and help him/her receive a broader view of the recommended items. Towards this direction, we exploit preferences defined by users upon items and extract a ranking of preferences that is used for ordering the suggested items. We further enrich this structure by summarizing the different levels of preferences with information for the items.

In a nutshell, this paper makes the following contributions:

- We propose a framework for time-aware recommendations that models the different types of time effects, that is, the age and the temporal context of ratings. Furthermore, we consider different cases for selecting the appropriate set of users for estimating the recommendations of a user and introduce the notion of support in recommendations to model how confident the recommendations of an item for a user is, in order to deal with the sparsity of the explicitly defined user ratings.
- We propose an effective presentation solution for the recommended items which builds upon user preferences for items. Our solution provides a ranked overview of the suggested items enriched with summarized information and can facilitate user browsing.

– We implement a proof of concept prototype for time-aware recommendations in a movie recommendations application and we experiment with different types of aging and temporal contexts. Our experiments show that time is an important dimension and should be part of the recommendation process.

The rest of the paper is organized as follows. The basic, time-invariant recommendation model is presented in Sect. 2. The time dimension is introduced in Sect. 3, where we distinguish between the aging factor (Sect. 3.1) and the temporal context factor (Sect. 3.2). In Sect. 4, we focus on the effective presentation of time-aware recommendations based on user preferences. The computation of recommendations under different temporal semantics is discussed in Sect. 5. In Sect. 6, we present our experiments using a real dataset of movie ratings. Our prototype implementation is outlined in Sect. 7, while related work is presented in Sect. 8. Finally, conclusions and outlook are pointed out in Sect. 9.

2 The Basic Time-Free Recommendation Model

Assume a set of items $\mathcal{I}$ with relational schema $R(A_1, \ldots, A_d)$, where each attribute A_j, $1 \leq j \leq d$, takes values from a domain $dom(A_j)$. Let $A = \{A_1, \ldots, A_d\}$ be the attribute set of R and $dom(A) = dom(A_1) \times \ldots \times dom(A_d)$ be its value domain. We use i to denote an item in $dom(A)$ of R. For instance, consider the movies shown in Fig. 1.

mID	*title*	*year*	*director*	*genre*	*language*	*duration*
1	Casablanca	1942	Curtiz	Drama	English	102
2	Vertigo	1958	Hitchcock	Horror	English	128
3	Psycho	1960	Hitchcock	Horror	English	109
4	Schindler's List	1993	Spielberg	Drama	English	195
5	The Farmer's Wife	1945	Hitchcock	Drama	English	129
6	Suspicion	1941	Hitchcock	Drama	English	99
7	Twilight Zone: The Movie	1983	Spielberg	Horror	English	101
8	Arachnophobia	1990	Spielberg	Horror	English	103
9	Lincoln	2012	Spielberg	Drama	English	150
10	The Walking Dead	1936	Curtiz	Horror	English	66

Fig. 1. Movies instance

Assume also a set of users $\mathcal{U}$. Each user $u \in \mathcal{U}$ may give a rating for an item $i \in \mathcal{I}$, which is denoted by $rating(u, i)$ and lies in the range $[0.0, 1.0]$. For instance, consider the ratings shown in Fig 2. We use $\mathcal{Z}_i$ to denote the set of users in $\mathcal{U}$ that have expressed a rating for item i. The cardinality of the items set $\mathcal{I}$ is usually high and typically users rate only a few of these items, that is, $|\mathcal{Z}_i| << |\mathcal{U}|$ for a specific item i. For the items unrated by the users, a *relevance score* is estimated by invoking a recommendation strategy.

uID	*mID*	*rating*	*timestamp*
1	3	0.9	1296367200
1	1	0.6	1297317600
2	2	0.7	1294639200
2	3	0.9	1298181600

Fig. 2. Ratings instance

In this section, we first present the basic model for time-free recommendations (Sect. 2.1) and then define the top-k recommendations problem (Sect. 2.2). The time–free recommendations model is the generally used recommendations model where the notion of time is completely ignored.

2.1 Defining Time-Free Recommendations

There are different ways to estimate the relevance of an item for a user. In general, the recommendation methods are organized into three main categories: (i) *content-based*, that recommend items similar to those the user has preferred in the past (e.g., [33,28]), (ii) *collaborative filtering*, that recommend items that similar users have liked in the past (e.g., [21,11]) and (iii) *hybrid*, that combine content-based and collaborative filtering approaches (e.g., [8]).

Our work falls into the *collaborative filtering* category. The key concept of collaborative filtering is to use, for a given user $u \in \mathcal{U}$, the ratings of other users in $\mathcal{U}$ in order to produce relevance scores for the items unrated by u. But, *which is the appropriate set of users, hereafter called peers, for computing the recommendations of u?* Due to the inherent fuzziness associated with this question, there exists no single definition for locating the peers of u. In our model, we consider three different aspects of peers: (i) *close friends*, (ii) *area experts* and (iii) *similar users*.

The *close friends* of a user u are explicitly selected by u. Computing recommendations using close friends is based on the assumption that these users would have similar tastes for most things, because of the closeness of the relationship.

Close Friends: Let $\mathcal{U}$ be a set of users. The *close friends* $\mathcal{C}_u$, $\mathcal{C}_u \subseteq \mathcal{U}$, of a user $u \in \mathcal{U}$ are explicitly defined by u.

An alternative solution might be the implicit extraction of the set of friends through some social network like Facebook or Google+.

From a different perspective, *area experts* can be used for producing recommendations for specific queries, since they are considered to be knowledgeable on a specific topic, domain or area. Several methods deal with the problem of finding experts (e.g., [9]); the focus of this paper though is on how to exploit experts preferences to recommend interesting items to other users and not on how to identify these experts. So, we consider that the set of experts for a given query are predefined, e.g., experts in tablet pcs.

Area Experts: Let $\mathcal{U}$ be a set of users and Q be a query. The *area experts* $\mathcal{D}_Q$, $\mathcal{D}_Q \subseteq \mathcal{U}$, are the users considered as experts for the query Q.

We denote this set as $\mathcal{D}_Q$, so, not dependent on the user, since typically experts are associated with specific queries, subjects or domains rather than with certain users.

Alternatively, a user can opt to employ the ratings of the users that exhibit the most similar behavior to him/her in order to produce relevance scores for the items unrated by him/her, even if other friendship or expert relationships exist. *Similar users* are located via a *similarity function* $simU(u, u')$ that evaluates the proximity between two users u and u'. Several methods can be applied for selecting the similar users of a user u. A direct method is to locate those users u' with similarity $simU(u, u')$ above a given threshold.

Similar Users: Let $\mathcal{U}$ be a set of users. The *similar users* $\mathcal{S}_u$, $\mathcal{S}_u \subseteq \mathcal{U}$, of a user $u \in \mathcal{U}$ is a set of users, such that, $\forall u' \in \mathcal{S}_u$, $simU(u, u') \geq \delta$ and $\forall u'' \in \mathcal{U} \backslash \mathcal{S}_u$, $simU(u, u'') < \delta$, where δ is a threshold similarity value.

Clearly, one could argue for other ways of selecting $\mathcal{S}_u$, e.g., by taking the k most similar users to u. Our main motivation here is that we opt for selecting only highly relevant users.

We define now the general notion of peers for a user by taking into account the three different cases presented above.

Definition 1 (Peers). *Let $\mathcal{U}$ be a set of users, u be a user in $\mathcal{U}$ and Q be a query posed by u. The peers $\mathcal{P}_{u,Q}$, $\mathcal{P}_{u,Q} \subseteq \mathcal{U}$, of u for Q are either:*

(i) the close friends $\mathcal{C}_u$ of u,
(ii) the area experts $\mathcal{D}_Q$ for Q, or
(iii) the similar users $\mathcal{S}_u$ of u.

Based on the peers of a user for a query, we formally define the relevance of an item for a user as follows:

Definition 2 (Time-free Relevance). *Let $\mathcal{U}$ be a set of users and $\mathcal{I}$ be a set of items. Let also Q be a query posed by $u \in \mathcal{U}$, and $\mathcal{P}_{u,Q}$ be the peers of u for Q. If u has not expressed any rating for an item $i \in \mathcal{I}$, the time-free relevance of i for u under Q is:*

$$relevance^f(u, i, Q) = \frac{\sum_{u' \in (\mathcal{P}_{u,Q} \cap \mathcal{Z}_i)} contribution(u, u') \times rating(u', i)}{\sum_{u' \in (\mathcal{P}_{u,Q} \cap \mathcal{Z}_i)} contribution(u, u')}$$

$$where\ contribution(u, u') = \begin{cases} 1, & if\ \mathcal{P}_{u,Q}\ is\ \mathcal{C}_u\ or\ \mathcal{D}_Q \\ simU(u, u'), & if\ \mathcal{P}_{u,Q}\ is\ \mathcal{S}_u \end{cases}$$

The relevance score of user u for an item i depends on the peers of u that have given a rating for i, i.e., those in $\mathcal{P}_{u,Q} \cap \mathcal{Z}_i$. The $contribution(u, u')$ reflects the importance of each $rating(u', i)$ for u; this importance depends on how "reliable" u' is for u. When close friends or area experts are used, contribution is set to 1, since we are certain about the importance of the ratings of the selected users

to the given user. For the similar users case, the contribution of each user u' depends on his/her similarity to u.

As already mentioned, due to the abundance of items in a recommendation application, users typically rate only a small portion of these items. So, the following question usually arises: *How confident are the relevance scores associated with the recommended items?* To deal with this issue, we introduce the notion of *support* for each candidate item i for user u, which defines the fraction of peers of u that have provided ratings for i.

Definition 3 (Time-free Support). *Let $\mathcal{U}$ be a set of users and $\mathcal{I}$ be a set of items. Let also Q be a query posed by $u \in \mathcal{U}$, and $\mathcal{P}_{u,Q}$ be the peers of u for Q. The time-free support of an item $i \in \mathcal{I}$ for u under Q is:*

$$support^f(u, i, Q) = |\mathcal{P}_{u,Q} \cap \mathcal{Z}_i| / |\mathcal{P}_{u,Q}|$$

Intuitively, the notion of support expresses how reliable is our estimation of the relevance of item i for user u.

To estimate the worthiness of an item recommendation for a user, we propose to combine the *relevance* and *support* scores in terms of a *value* function.

Definition 4 (Time-free Value). *Let $\mathcal{U}$ be a set of users and $\mathcal{I}$ be a set of items. For $\sigma \in [0, 1]$, the time-free value of an item $i \in \mathcal{I}$ for a user $u \in \mathcal{U}$ under a query Q, such that, $\nexists rating(u, i)$, is:*

$$value^f(u, i, Q) = \sigma \times relevance^f(u, i, Q) + (1 - \sigma) \times support^f(u, i, Q)$$

We take a generic approach for computing the *time-free value* of an item for a user. More sophisticated functions can be designed. However, this linear combination of relevance and support is simple and easy to implement. Moreover, when $\sigma = 1$, *value* maps to *relevance*, which is the typically used recommendation score.

2.2 Top-k Time-Free Recommendations

Given a query Q submitted by a user u and a restriction k on the number of the recommended items, the goal is to provide u with k suggestions for items that are highly relevant to u and exhibit high support.

Definition 5 (Top-k Time-free Recommendations). *Let $\mathcal{U}$ be a set of users and $\mathcal{I}$ be a set of items. Given a query Q posed by a user $u \in \mathcal{U}$, recommend to u a list of k items $\mathcal{I}_u =< i_1, \ldots, i_k >$, $\mathcal{I}_u \subseteq \mathcal{I}$, such that:*
(i) $\forall i_j \in \mathcal{I}_u$, $\nexists rating(u, i_j)$,
(ii) $value^f(u, i_j, Q) \geq value^f(u, i_{j+1}, Q)$, $1 \leq j \leq k-1$, $\forall i_j \in \mathcal{I}_u$, *and*
(iii) $value^f(u, i_j, Q) \geq value^f(u, x_y, Q)$, $\forall i_j \in \mathcal{I}_u$, $x_y \in \mathcal{I} \backslash \mathcal{I}_u$.

The first condition ensures that the suggested items do not include already evaluated items by the user (for example, do not recommend a movie that the user has already watched). The second condition ensures the descending ordering of the items with respect to their value, while the third condition defines that every item in the result set has value greater than or equal to the value of any of the non–suggested items.

3 Time-Aware Recommendations

The basic time-free recommendation model presented above assumes that all ratings are active and potentially they could be exploited for recommendations. This way though the temporal aspects of the user ratings are completely ignored. However, the information needs of a user evolve over time, especially if we consider a long period of time, either smoothly (i.e., drift) or more drastically (i.e., shift). As such, the recent user ratings reflect better his/her current interests comparing to older possible obsolete ratings. From another point of view, user interests might change under different temporal circumstances and thus, users may have different needs depending on the temporal context. For example, during the weekdays one might be interested in reading IT news whereas during the weekends he/she might be interested in reading about cooking, gardening or other hobbies.

To handle such different cases, we propose a framework for time-aware recommendations that incorporates the notion of time in the recommendation process towards accuracy improvement. We distinguish between two types of time-aware recommendations, namely the fresh-based and the context-based ones. The *fresh-based recommendations* pay more attention to more recent user ratings thus trying to deal with the problem of drift or shift in the user information needs over time. The *context-based recommendations* take into account the temporal context under which the ratings were given (e.g., weekdays vs weekends).

In our time-aware recommendation model, the rating of a user u for an item i, $rating(u, i)$, is associated with a timestamp $t_{u,i}$, which is the time that i was rated by u (c.f., Fig 2) and thus, it denotes the freshness or age of the rating. Below, we first define the fresh-based recommendation model (Sect. 3.1) and then, the temporal context-based recommendation model (Sect. 3.2). We also present a variant of the top-k recommendations problem by defining the top-k time-aware recommendations (Sect. 3.3).

3.1 Fresh-Based Recommendations

Generally speaking, the popularity of the items in a recommendation application changes over time; typically, items, e.g., movies, pictures or songs, lose popularity as time goes by. Motivated by the intuition that the importance of item ratings increases with the popularity of the items themselves, fresh-based recommendations suggest items by mainly exploiting recent and novel user ratings.

Driven by the work in data streams [18], we use different types of aging mechanisms to define the way that the historical information (in form of ratings) is incorporated in the recommendation process. Aging in streams is typically implemented through the notion of windows, which define which part of the stream is active at each time point and thus could be used for further processing. In this work, we use the *damped window model* that gradually decreases the importance of historical data comparing to more recent data and the *sliding window model* that remembers only the ratings given within a specific, recent time period. We present these cases in more detail below. Note that the static case (Sect. 2),

corresponds to the *landmark window model* which considers the whole rating history from a given landmark.

Damped Window Model. In the damped window model, although all user ratings are active, i.e., they can contribute to produce recommendations, their contribution depends upon their arrival time, i.e., upon the time of rating. In particular, the rating of a user u for an item i is weighted through some temporal decay function that gradually discounts the history of past ratings. Typically, in temporal applications, the exponential fading function is employed, so the weight of $rating(u, i)$ decreases exponentially with time via the function $2^{-\lambda(t-t_{u,i})}$, where $t_{u,i}$ is the time of the rating and t is the current time. Thus, $t - t_{u,i}$ is the age of the rating. The parameter λ, $\lambda > 0$, is the decay rate which defines how fast the past history is forgotten. The higher λ, the lower the importance of historical ratings compared to more recent ratings.

Under this aging schema, the so-called *damped relevance* of an item i for a user u with respect to a query Q in a given timepoint t is given by:

$$relevance^d(u, i, Q) = \frac{\sum_{u' \in (\mathcal{P}_{u,Q} \cap \mathcal{Z}_i)} 2^{-\lambda(t-t_{u',i})} \times contribution(u, u') \times rating(u', i)}{\sum_{u' \in (\mathcal{P}_{u,Q} \cap \mathcal{Z}_i)} contribution(u, u')}$$

So, all user item scores $rating(u', i)$ are weighted by their recency $2^{-\lambda(t-t_{u',i})}$.

Since all ratings are active, the *damped support* of i for u under Q is equal to the corresponding time-free support, that is:

$$support^d(u, i, Q) = support^f(u, i, Q)$$

Finally, the *damped value* of i for u under Q is computed as in the time-free case by combining the relevance and support scores ($\sigma \in [0, 1]$):

$$value^d(u, i, Q) = \sigma \times relevance^d(u, i, Q) + (1 - \sigma) \times support^d(u, i, Q)$$

Sliding Window Model. In the sliding window model only a subset of the available ratings is exploited, and in particular, the most recent ones. The size of this subset, referred to as window size, might be defined in terms of timepoints (e.g., use the ratings given within the last month) or records (e.g., use the 1000 most recent ratings). We adopt the first case. The ratings within the window are the active ratings that participate in the recommendation computation. Let t be the current time and W be the window size. Then, a rating of a user u for an item i, $rating(u, i)$, is active only if $t_{u,i} > t - W$.

In the sliding window model, the *sliding relevance* of an item i for a user u under a query Q is defined with regard to the active ratings of the peers of u for i. More specifically:

$$relevance^s(u, i, Q) = \frac{\sum_{u' \in (\mathcal{P}_{u,Q} \cap \mathcal{X}_i)} contribution(u, u') \times rating(u', i)}{\sum_{u' \in (\mathcal{P}_{u,Q} \cap \mathcal{X}_i)} contribution(u, u')}$$

where $\mathcal{X}_i$ is the set of users in $\mathcal{Z}_i$, such that, $\forall u' \in \mathcal{X}_i$, $t_{u',i} > t - W$.

The *sliding support* of i for u under Q is defined as the fraction of peers of u that have expressed ratings for i that are active at time t. That is:

$$support^s(u, i, Q) = |\mathcal{P}_{u,Q} \cap \mathcal{X}_i| / |\mathcal{P}_{u,Q}|$$

Finally, the *sliding value* of i for u under Q, for $\sigma \in [0, 1]$, is a linear combination of their relevance and support scores:

$$value^s(u, i, Q) = \sigma \times relevance^s(u, i, Q) + (1 - \sigma) \times support^s(u, i, Q)$$

3.2 Temporal Context-Based Recommendations

In contrast to fresh-based recommendations, the context-based ones assume that although the user preferences may change over time, they display some kind of temporal repetition. Or in other words, users may have different preferences under different temporal contexts. For instance, during the weekend a user may prefer to watch different movies from those in the weekdays. So, a movie recommendation system should provide movie suggestions for the weekends that may differ from the suggestions referring to weekdays.

As above, the rating of a user for an item, $rating(u, i)$, is associated with the rating time $t_{u,i}$. Time is modeled here as a multidimensional attribute. The dimensions of time have a hierarchical structure, that is, time values are organized at different levels of granularity (similar to [32,35]). In particular, we consider three different levels over time: *time_of_day*, *day_of_week* and *time_of_week* with domain values {"morning", "afternoon", "evening", "night"}, {"Mon", "Tue", "Wed", "Thu", "Fri", "Sat", "Sun"} and {"Weekday", "Weekend"}, respectively. It is easy to derive such kind of information from the time value $t_{u,i}$ that is associated with each user rating by using SQL or other programming languages. More elaborate information can be extracted by using the WordNet or other ontologies.

Let Θ be the current temporal context of a user u. We define the *context-based relevance* of an item i for u under a query Q expressed at Θ based on the ratings of the peers of u for i that are defined for the same context Θ. Formally:

$$relevance^c(u, i, Q) = \frac{\sum_{u' \in (\mathcal{P}_{u,Q} \cap \mathcal{Y}_i)} contribution(u, u') \times rating(u', i)}{\sum_{u' \in (\mathcal{P}_{u,Q} \cap \mathcal{Y}_i)} contribution(u, u')}$$

where $\mathcal{Y}_i$ is the set of users in $\mathcal{Z}_i$, such that, $\forall u' \in \mathcal{Y}_i$, $t_{u',i} \mapsto \Theta$, that is, the user rating has been expressed for a context equal to Θ. For example, if the temporal context of a user query is "Weekend", only the user ratings given for the context "Weekend" would be considered.

The *context-based support* of i for u under Q is defined with respect to the number of peers of u that have expressed ratings for i under the same temporal context as the query context. That is:

$$support^c(u, i, Q) = |\mathcal{P}_{u,Q} \cap \mathcal{Y}_i| / |\mathcal{P}_{u,Q}|$$

Similar to the fresh-based recommendations, the *context-based value* of i for u under Q is calculated taking into account the context-based relevance and support. For $\sigma \in [0, 1]$:

$$value^c(u, i, Q) = \sigma \times relevance^c(u, i, Q) + (1 - \sigma) \times support^c(u, i, Q)$$

3.3 Top-k Time-Aware Recommendations

Next, we define the time-aware variation of the top-k recommendation problem (c.f., Sec 2.2) applicable to both fresh-based and context-based approaches.

Definition 6 (Top-k Time-aware Recommendations). *Let $\mathcal{U}$ be a set of users and $\mathcal{I}$ be a set of items. Given a query Q posed by a user $u \in \mathcal{U}$ at time t mapped to a temporal context Θ, recommend to u a list of k items $\mathcal{I}_u =< i_1, \ldots, i_k >$, $\mathcal{I}_u \subseteq \mathcal{I}$, such that:*

(i) $\forall i_j \in \mathcal{I}_u$, $\nexists rating(u, i_j)$, *for the fresh-based recommendations, and* $\forall i_j \in \mathcal{I}_u$, $\nexists rating(u, i_j)$ *that is associated with context equal to Θ, for the context-based recommendations,*

(ii) $value^o(u, i_j, Q) \geq value^o(u, i_{j+1}, Q)$, $1 \leq j \leq k-1$, $\forall i_j \in \mathcal{I}_u$, *and*

(iii) $value^o(u, i_j, Q) \geq value^o(u, x_y, Q)$, $\forall i_j \in \mathcal{I}_u$, $x_y \in \mathcal{I} \backslash \mathcal{I}_u$,

where o corresponds to either d (for the damped window model), s (for the sliding window model) or c (for the context-based model).

The first condition ensures that the suggested items do not include already evaluated items by the user either in general or under a specific context, while the second and the third conditions resemble those of Def. 5.

4 Presentation of Time-Aware Recommendations Based on User Preferences

Depending on the value of k and the recommendation application per se, the top-k recommendations for a user u might result in a lot of information for u. To facilitate the user selection, we propose to organize the results in a compact yet intuitive and representative way. To achieve this goal, we employ, apart from ratings, preferences expressed by users over items. These user preferences might be either qualitative (e.g., the director is more important than the genre of the movie) or quantitative (e.g., the preference scores for the directors Q. Tarantino, F. F. Coppola are 0.9, 0.5, respectively).

In the following, we discuss in more details how preferences can be given (Sect. 4.1) and we present a formal model for the effective presentation of user top-k recommendations based on his/her preferences (Sect. 4.2).

4.1 User Preferences

In general, preferences can be expressed either in a qualitative or in a quantitative way. Following a qualitative preference model, users employ binary relations to directly define preferences between data items (e.g., [13,20]). Following a quantitative preference model, users provide numeric scores via scoring functions to indicate their degree of interest (e.g., [6,22,35]). We use a qualitative preference model [13], since this model is more general than the quantitative one and also closer to the users intuition. Specifically:

Definition 7 (Preference Model). *Let $\mathcal{U}$ be a set of users and $\mathcal{I}$ be a set of items with relational schema $R(A_1, \ldots, A_d)$. For a user $u \in \mathcal{U}$, assume a set of values P_u of an attribute A_j, $1 \leq j \leq d$, such that, $P_u \subseteq dom(A_j)$. The user u specifies a binary preference relation $pref_u$ on P_u, $pref_u = \{(p_1 \succ p_2) | p_1, p_2 \in P_u\}$, where $p_1 \succ p_2$ denotes that u prefers p_1 over p_2.*

For example, a user might prefer A. Hitchcock over S. Spielberg, i.e., *(A. Hitchcock $\succ$ S. Spielberg)*, and S. Spielberg over Q. Tarantino, i.e., *(S. Spielberg $\succ$ Q. Tarantino)*.

Alternatively, instead of providing comparative relationships, users could provide explicit preference scores for the values in P_u. This would correspond to a quantitative approach, with higher preference scores indicating more important preferences. For example, a user might assign to A. Hitchcock, S. Spielberg and Q. Tarantino the preference scores 0.9, 0.7 and 0.6, respectively. The transition from the quantitative to the qualitative approach is straightforward. For the aforementioned example, the qualitative equivalent is: *(A. Hitchcock $\succ$ S. Spielberg)*, *(A. Hitchcock $\succ$ Q. Tarantino)*, and *(S. Spielberg $\succ$ Q. Tarantino)*.

Irrespectively of their qualitative or quantitative formulation, preferences may also be expressed at different levels of granularity. We distinguish between value-based and attribute-based preferences. *Value-based preferences* are expressed between individual values of item attributes. Typically, they are formulated over the items of a relation based on the values of their attributes. An example of a value-based preference for the item attribute director could be *(A. Hitchcock $\succ$ Q. Tarantino)*. *Attribute preferences* express preferences between the different attributes of R, i.e., they evaluate how important for the end user each attribute or feature of the item description is. For example, a user might consider the attribute director more important than the attribute genre, i.e., *(director $\succ$ genre)*. Attribute preferences might be also expressed either qualitatively (e.g., [19]) or quantitatively (e.g., [26]). In what follows, we will use the term $pref_u$ to denote the whole set of value-based and attribute-based preferences of a user u.

Clearly, preferences may be collected using various ways. Specifically, preferences can be provided explicitly by the users, as above, or constructed automatically, for instance, based on the past behavior of the user or of similar users. Such methods for the automatic construction of preferences have been the focus of much current research (e.g., [27]) and are beyond the scope of this paper. For our study, we assume that the set of preferences is provided for each user.

4.2 Time-Aware Recommendations Presentation

A user in a recommendation application might express both value-based and attribute-based preferences. To combine these different types, we use attribute-based preferences to set priorities among value-based preferences based on the attributes involved in the preferences, similarly to [19]. For example, assume a user with value-based preference *(A. Hitchcock* $\succ$ *S. Spielberg)* over the attribute director and *(horror* $\succ$ *drama)* over the attribute genre. Assume also that our user considers the director of a movie to be more important than its genre which is expressed through the attribute-based preference *(director* $\succ$ *genre).*

Given the above set of preferences, the following combined preferences can be drawn: our user prefers the set of values or keywords {*A. Hitchcock, horror*} over the set of values {*A. Hitchcock, drama*}. The latter set is preferred over the set {*S. Spielberg, horror*}, which in turn is preferred over the set {*S. Spielberg, drama*}.

The combined preferences of a user can be directly exploited for ranking the top-k recommendations of the user. The idea is to first extract the combined preferences and then use them to rank and present the top-k recommended items to the user.

This ranking could be also enhanced by exploiting the different attributes of the item description and building summaries upon these descriptions. We start with the brick of this concept, which is, the keyword-based summary.

Definition 8 (Keyword-based summary). *Let M be a set of keywords and $\mathcal{I}'$, $\mathcal{I}' \subseteq \mathcal{I}$, be a set of items. A keyword-based summary, key-sum, is a pair (M: $\mathcal{I}'_M$), $\mathcal{I}'_M \subseteq \mathcal{I}'$, such that, all items in $\mathcal{I}'_M$ contain all keywords in M.*

For example, the keyword-based summary *({A. Hitchcock, horror}: {Psycho, Vertigo})* consists of a set of two keywords *(A. Hitchcock* and *horror)* that is associated with a set of movies *(Psycho* and *Vertigo)* that contain (or are related with) the keywords. In this example, the keywords {*A. Hitchcock, horror*} derived from user preferences are extended by the titles of the movies which are directed by A. Hitchcock and belong to the horror genre type. Other extension options could be employed as well, e.g., information about the production year or the duration of the movies. Also, the extension might refer to more than one attributes, e.g., both title and production year of a movie could be considered. Such a summary offers more information to the end user regarding the recommended item and facilitates his/her selection.

So far, we focus on the summary of a single combined preference. Our goal is to construct a summary for the top-k recommendations based on user preferences. This summary consists of an ordered set of keyword-based summaries, such that, a keyword-based summary $key\text{-}sum_i = (M_i, \mathcal{I}'_{M_i})$ appears before a keyword-based summary $key\text{-}sum_j = (M_j, \mathcal{I}'_{M_j})$, if the keywords of M_i are preferred over the keywords of M_j with respect to the available value-based and attribute-based preferences. Considering our previous example and using only the titles of the movies for the extension, the corresponding ordering would be: the summary *({A. Hitchcock, horror}: {Psycho, Vertigo})* is preferred over the summary

({A. Hitchcock, drama}: {The Farmer's Wife, Suspicion}), which is preferred over the summary *({S. Spielberg, horror}: {Twilight Zone: The Movie, Arachnophobia})*, which in turn is preferred over *({S. Spielberg, drama}: {Lincoln, Schindler's List})*.

Note also that there might be cases where the preferences may be equivalent, e.g., {*M. Curtiz, horror*} is equally preferred to {*S. Spielberg, horror*}, and consequently, the corresponding keyword-based summaries would be equivalent. To accommodate such cases, we propose to summarize the equivalent keyword-based summaries in the so called keyword-based class summaries.

Definition 9 (Keyword-based class summary). *A keyword-based class summary, class-sum, is a set of keyword-based summaries {key-sum$_1$, ..., key-sum$_n$}, such that, the keywords in M_1, ..., M_n are considered equally preferable with respect to a given set of value and attribute preferences.*

For example, given that the sets of keywords {*M. Curtiz, horror*} and {*S. Spielberg, horror*} are equally preferable with respect to a specific set of preferences, then their keyword-based summaries, e.g., {*({M. Curtiz, horror}: {The Walking Dead})* and *({S. Spielberg, horror}: {Twilight Zone: The Movie, Arachnophobia})*}, constitute a keyword-based class summary.

Based on the keyword-based class summaries, we define formally the time-aware recommendations summary as follows:

Definition 10 (Time-aware recommendations summary). *Let $\mathcal{U}$ be a set of users, $\mathcal{I}$ be a set of items, Q be a query posed by a user $u \in \mathcal{U}$ at time t mapped to the temporal context Θ and $pref_u$ be the set of value and attribute preferences of u. Let also $\mathcal{I}_u$ be the top-k time-aware recommendations for u. The time-aware recommendation summary for u is a list of keyword-based class summaries tar-sum = <class-sum$_1$, ..., class-sum$_x$>, such that:*

(i) all sets of keywords in class-sum$_i$ are preferred over all sets of keywords in class-sum$_{i+1}$, $1 \leq i \leq x-1$, with respect to $pref_u$,
(ii) all keywords of the value preferences of $pref_u$ appear in an M set of tar-sum and
(iii) only the keywords of the M sets of tar-sum appear in the value preferences of $pref_u$.

So, given that the set of keywords {*A. Hitchcock, horror*} is preferred over the sets {*M. Curtiz, horror*} and {*S. Spielberg, horror*}, with the last two being

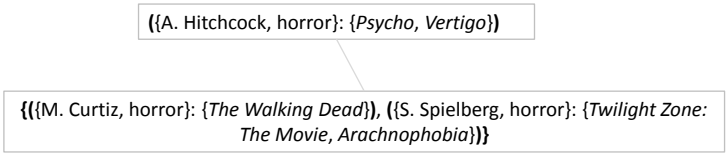


Fig. 3. A presentation example

equally preferred, then the keyword-based summary for {*A. Hitchcock, horror*} will be first displayed. The keyword-based class summary for {*M. Curtiz, horror*} and {*S. Spielberg, horror*} will follow. Schematically, this would look as in Fig. 3.

5 Time-Aware Recommendations Computation

A high level representation of the main components of the architecture of our system is depicted in Fig. 4. Assume a user that submits a query presenting his information needs. Each query is enhanced with a contextual specification expressing some temporal information. This temporal information of the query may be postulated by the application or be explicitly provided by the user as part of his query. Typically, in the first case, the context implicitly associated with a query corresponds to the current context, that is, the time of the submission of the query. As a query example, for a restaurant recommendation application, consider a user looking for restaurants serving chinese cuisine during the weekend. As part of his/her query, the user should also provide the aging schema that will be used.

Then, we locate the peers of the user (Sect. 5.1) and employ their ratings for estimating the time-aware recommendations (Sect. 5.2). Finally, recommendations are summarized and presented to the user (Sect. 5.3). In following, we overview the details of each step.

5.1 Selecting Peers

Our model assumes three different kinds of peers, namely close friends, area experts and similar users. For each submitted query Q of a user u, u specifies the peers that will be used for producing his/her recommendations. This selection step of the peers is, in general, application dependent. For example, when a user is asking for advice for a personal computer, the area experts may fit well to the user needs, while when asking for a suggestion about a movie, the user's close friends may provide good answers. In a similar manner, when using a trip advisor, the choice of users with similar tastes seems appropriate.

For the close friends case, the set of peers of u consists of the close friends of u, while for the area experts case, the set of peers of u consists of the users that are considered to be experts for Q. We assume that this information is already known. For the similar users case, we need to calculate all similarity measures $simU(u, u')$ for all users $u' \in \mathcal{U}$. Those users u' with similarity $simU(u, u')$ greater than or equal to the threshold δ represent the similar users of u (Algorithm 1).

5.2 Computing Recommendations

Having established the methodology for finding the peers of a user, we focus next on how to generate valued recommendations for him/her. Given a user $u \in \mathcal{U}$ and his/her peers $\mathcal{P}_{u,Q}$, the procedure for estimating the value score of an item

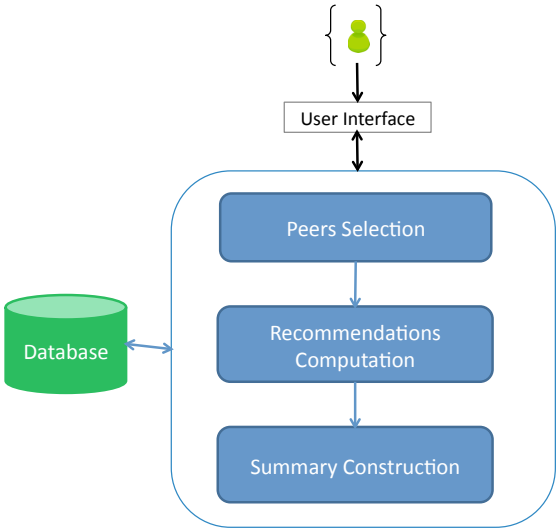


Fig. 4. System architecture

i for u requires the computation of the relevance and support of i. Note that we do not compute value scores for all items in $\mathcal{I}$, but only for the items $\mathcal{I}'$, $\mathcal{I}' \subseteq \mathcal{I}$, that satisfy the query selection conditions. To do this, we perform a pre-processing step to select the relevant to the query data by running a typical database query. For example, for a query about destinations in Greece posed to a travel recommendation system, we ignore all the rest destinations.

Algorithm 2 presents the general procedure for computing the value scores of the items in $\mathcal{I}'$. Pairs of the form $(i, value^o(u, i, Q))$ are maintained in a set $\mathcal{V}_u$, where o corresponds to d, s or c for the damped window, sliding window and context-based approach, respectively. As a post-processing step, we rank all pairs in $\mathcal{V}_u$ on the basis of their value score. To provide the top-k recommendations to u, we report the k items with the highest scores, i.e., the k first items in $\mathcal{V}_u$.

Next, we discuss separately the particulars of each time-aware recommendation approach. For the damped window approach, all the ratings of the peers of u are employed for computing recommendations. However, this is not the case for the other two approaches, where only a subset of the peers ratings are taken into consideration. More specifically, for the sliding window approach, only the most recent ratings are used, while for the context-based approach, the ratings that are defined for a temporal context equal to the query context are employed. This can be seen as a rating pre-filtering step. It is worth noting that, since some ratings are ignored due to temporal specifications, some of the peers finally may not contribute at all to the recommendation list construction.

Moreover, for the context-based approach, the associated set of ratings for a specific query may be empty, that is, there may be no ratings for the query. In this case, we can use for the recommendation process these ratings whose context is more general than the query context. For example, for a query with

Algorithm 1. Finding Similar Users Algorithm

Input: A set of users $\mathcal{U}$, a user $u \in \mathcal{U}$ and a threshold similarity value δ.
Output: The peers of u, $\mathcal{P}_{u,Q}$.

```
1: begin
2: P_{u,Q} = ∅;
3: for each user u' ∈ U\{u} do
4:    compute simU(u, u');
5:    if simU(u, u') ≥ δ then
6:       add u' to P_{u,Q};
7:    end if
8: end for
9: return P_{u,Q};
10: end
```

Algorithm 2. Value Computation Algorithm

Input: A user $u \in \mathcal{U}$, a query Q, the peers $\mathcal{P}_{u,Q}$ along with their ratings, the aging schema and the weight σ.
Output: A set $\mathcal{V}_u$ of pairs $(i, value^o(u, i, Q))$, $\forall i \in \mathcal{I}'$.

```
1: begin
2: V_u = ∅;
3: for each item i ∈ I' unrated by user u do
4:    compute relevance^o(u, i, Q);
5:    compute support^o(u, i, Q);
6:    compute value^o(u, i, Q);
7:    add (i, value^o(u, i, Q)) to V_u;
8: end for
9: return V_u;
10: end
```

context "Sat", we can use a rating given for context "Weekend". The selection of the appropriate ratings can be made more efficient by deploying indexes on the context of the ratings. Such a data structure that exploits the hierarchical nature of context, termed profile tree, is introduced in [35].

As a final note, the two approaches for computing time-aware recommendations can be combined. For instance, we can apply the context-based approach first. Then, we can apply the damped window approach. This way, the importance of the ratings that are defined for the query context decreases with time.

5.3 Presenting Recommendations

To facilitate users in item selection, we present the top-k time-aware recommended items for a user u in a compact and intuitive way, by employing his/her value-based and attribute-based preferences $pref_u$.

The problem we deal with here can be stated as follows: *Given a relation R describing the items in a recommendation application and the preferences $pref_u$ of u over R, how to produce ranked groups of items in R based on $pref_u$.* To this

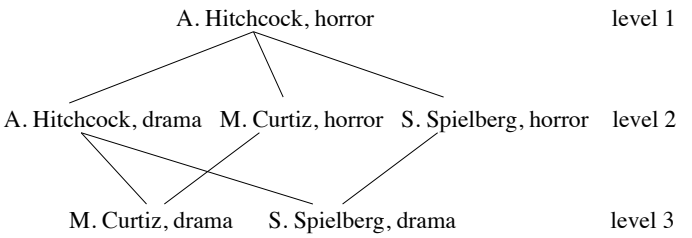


Fig. 5. A lattice example

end, a lattice is built where the nodes correspond to the combinations of values appearing in $pref_u$. For example, for the list of preferences: (i) A. Hitchcock is preferred over M. Curtiz or S. Spielberg, (ii) horror movies are preferred over drama movies and (iii) the director of a movie is as important as its genre, the lattice of Fig. 5 is constructed. The top nodes are more important to the user comparing to the bottom nodes, whereas nodes lying in the same level of the lattice are of equal importance.

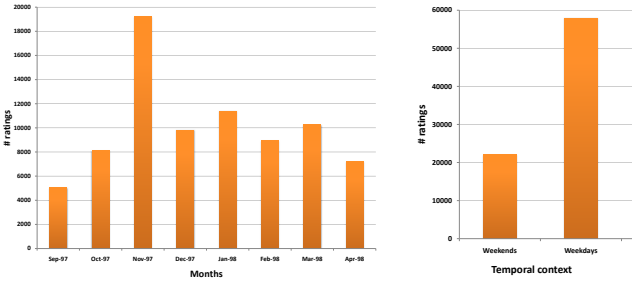
A query is formulated for each node in the lattice. Considering only the top-k items for recommendations, all queries in a specific level are associated with equally preferable items and each query is associated with the items that contain the keywords of the query. The queries of each level are successively executed starting from the queries of the top level and going down the lattice. For example, for the lattice of Fig. 5, items with keywords {*A. Hitchcock, horror*}, i.e., items in the result of the query of the first level, are preferred over the items with keywords {*S. Spielberg, horror*}, i.e., the items in the results of a query of the second level, and so on. Within the same level, items are ranked according to their recommendation value score.

Then, we construct a keyword-based summary for each query in the lattice. The set of queries in a level of the lattice corresponds to a keyword-based class summary, while the total set of ordered queries in the lattice represents the time-aware recommendation summary.

6 Experiments

In this section, we evaluate the effectiveness of our time-aware recommendation system using a real movie ratings dataset [1], which consists of 100,000 ratings given from September 1997 till April 1998 by 1,000 users for 1,700 items. The monthly split is shown in Fig. 6(a), while the split per weekends and weekdays is shown in Fig. 6(b).

Since there is no information about actual friends and experts in the dataset, we employ as the peers of a given user his/her similar users. To this end, the notion of user similarity is important. We use here a simple variation; that is, we use distance instead of similarity. More specifically, we define the distance between two users as the Euclidean distance over the items rated by both.



(a) (b)

Fig. 6. (a) Ratings per month and (b) ratings per temporal context

Let $u, u' \in \mathcal{U}$ be two users, $\mathcal{I}_u$ be the set of items for which $\exists rating(u, i)$, $\forall i \in \mathcal{I}_u$, and $\mathcal{I}_{u'}$ be the set of items for which $\exists rating(u', i)$, $\forall i \in \mathcal{I}_{u'}$. We denote by $\mathcal{I}_u \cap \mathcal{I}_{u'}$ the set of items for which both users have expressed preferences. Then, the distance between u, u' is:

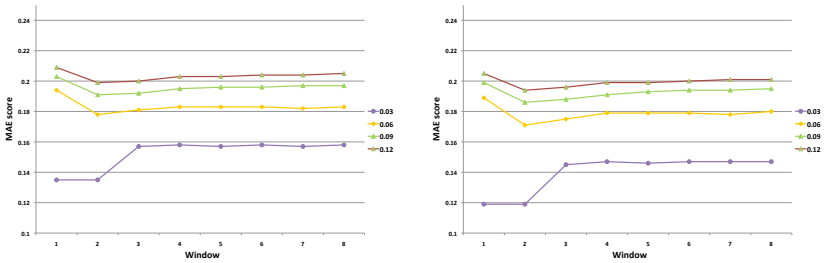
$$distU(u, u') = \sqrt{\sum_{i \in \mathcal{I}_u \cap \mathcal{I}_{u'}} (rating(u, i) - rating(u', i))^2} / |\mathcal{I}_u \cap \mathcal{I}_{u'}|$$

To evaluate the quality of the recommendations, we use a predictive accuracy metric that directly compares the predicted ratings with the actual ones [25]. A commonly used metric in the literature is the *Mean Absolute Error* (MAE), which is defined as the average absolute difference between predicted ratings and actual ratings: $MAE = \sum_{u,i} |rating(u, i) - value^o(u, i, Q)|/N$, where N is the total number of ratings in the employed dataset and o corresponds to d, s or c. Clearly, the lower the MAE score, the better the predictions.

Next, we report on the results for the sliding window model, the damped window model and the context-based model compared to the time-free model.

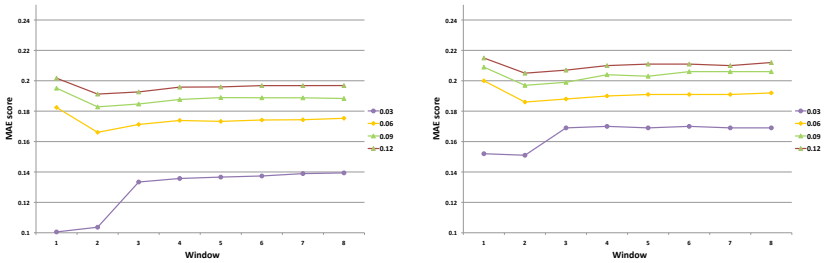
Sliding window model. To illustrate the effectiveness of the sliding window model, we use windows of different sizes W. The window size $W = 1$ stands for the most recent month, i.e., April 1998, the window size $W = 2$ stands for both April 1998 and March 1998, and so forth. The window size $W = 8$ includes the whole dataset, from April 1998 till September 1997. We denote the resulting dataset as D_W, where $W = [1 - 8]$ is the window size. For each dataset D_W, we compute the recommendations for each user by considering the user ratings within the corresponding window W. We compare the predicted values with the actual values given by the user within the same window W and report the average results.

The results for different windows, distance thresholds and σ values are presented in Fig. 7(a) (for $\sigma = 1.0$), Fig. 7(b) (for $\sigma = 0.95$), Fig. 8(a) (for $\sigma = 0.9$), Fig. 8(b) (for $\sigma = 0.85$), Fig. 9(a) (for $\sigma = 0.8$) and Fig. 9(b) (for $\sigma = 0.75$). Regarding the effect of the different window sizes, in general, recommendations present better quality for small windows (this is not the case for the smallest window size $W = 1$ because of the small amount of ratings used for predictions). For example, for a user distance threshold equal to 0.03 and $W = 3$,



(a) (b)

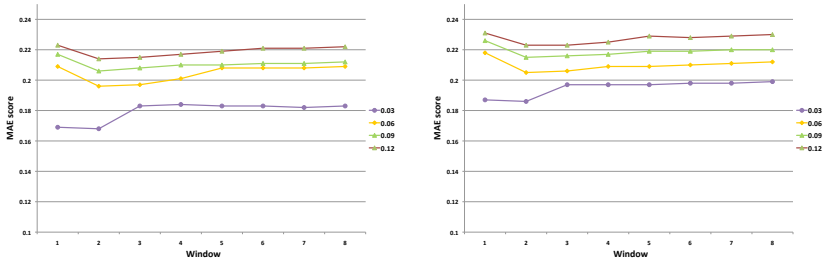
Fig. 7. MAE scores for the sliding window model with (a) $\sigma = 1.0$ and (b) $\sigma = 0.95$



(a) (b)

Fig. 8. MAE scores for the sliding window model with (a) $\sigma = 0.9$ and (b) $\sigma = 0.85$

the predictions are improved around 2.5% compared to $W = 8$ (i.e., compared to the time-free recommendations model). Or, for a threshold equal to 0.06 and $W = 2$, the predictions are improved around 4% compared to $W = 8$ (Fig. 7(a)). Moreover, the larger the window, the smaller the improvement. As expected, for larger user distance thresholds, the MAE scores increase for all window sizes, since more dissimilar users are considered for the suggestions computation. Note that the specific distance threshold values are selected with respect to the numbers of similar users they return; the experiment presents similar behavior for different such values. Regarding the effect of σ, the best recommendation quality is given when $\sigma = 0.9$. More specifically, the quality is improved from $\sigma = 1.0$ to $\sigma = 0.9$, while we notice the opposite behavior for smaller σ values. That is, for $\sigma \in [0.9, 1.0]$, the lower the σ, the better the predictions, and, for $\sigma < 0.9$, the higher the σ, the better the predictions. For example, for $W = 2$ and distance equal to 0.06, the predictions for $\sigma = 0.9$ are improved around 7% compared to the predictions for $\sigma = 1.0$ and 16% for $\sigma = 0.8$. The corresponding improvements for distance equal to 0.09 are 4.5% and 12%, respectively. In overall, our studies show that support has an effect on the recommendations quality and could be used for improving the recommendation process. However, the choice of an optimal value for σ to achieve the highest quality of recommendations is



(a) (b)

Fig. 9. MAE scores for the sliding window model with (a) $\sigma = 0.8$ and (b) $\sigma = 0.75$

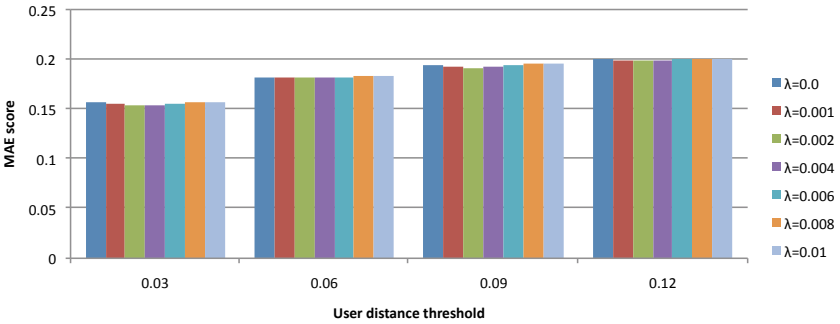


Fig. 10. MAE scores for the damped window model

application dependent, due to the different amounts of ratings given at specific time instances, or periods.

Damped window model. Next, we evaluate the effect of the decay rate λ in the recommendations accuracy. We use different values for λ; the higher the λ is, the less the historical data count. The value $\lambda = 0$ corresponds to the time-free model. We downgrade the original ratings based on the decay factor λ and the time difference between the end of the observation period (22/04/1998) and the ratings timestamp.

The results of this experiment for $\sigma = 0.9$, which is the optimal σ value according to the previous analysis, are shown in Fig. 10. This aging model offers a small improvement in this setting, i.e., for the employed dataset. In particular, the best MAE scores are obtained when $\lambda = 0.004$. So, for $\lambda = 0.004$ and distance equal to 0.03, the predictions are improved around 1.3% compared to the time-free model. The improvements for distance equal to 0.06, 0.09 and 0.12 are 0.6%, 1.6% and 1%, respectively. Larger λ values lead to worst predictions compared to the predictions of the time-free model. Practically, $\lambda = 0.004$ means

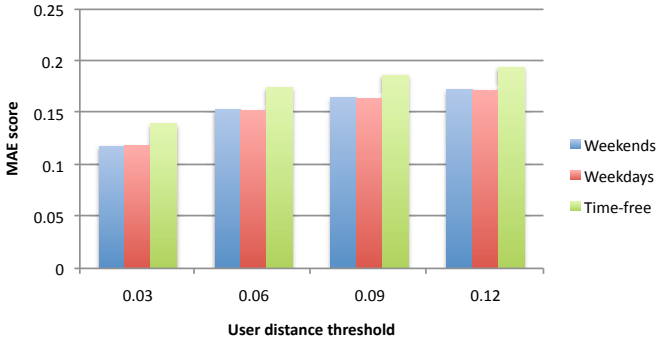


Fig. 11. MAE scores for the context-based approach

that the ratings loose 10% of their value after 10 years. As above, larger distance thresholds lead to larger MAE scores and worst recommendations quality.

Context-based recommendations. In this set of experiments, we demonstrate the effect of temporal context on producing recommendations. We consider two different temporal contexts "Weekends" and "Weekdays". For the "Weekends" context, we base our predictions only on ratings defined for weekends ($D_{weekends}$), whereas for the "Weekdays" context, we consider ratings from Monday to Friday ($D_{weekdays}$). The predicted values are compared to the actual values given by the user within the same temporal context through the MAE metric.

Fig. 11 displays the results for $\sigma = 0.9$. Except for the two temporal contexts, "Weekends" and "Weekdays", we also present the scores for the time-free model, i.e., when the whole dataset is used. Generally speaking, the temporal context affects the recommendations accuracy. In particular, for both contexts, "Weekends" and "Weekdays", the quality of the recommendations is improved compared to the time-free approach that completely ignores the temporal information of the ratings. For example, for a user distance threshold equal to 0.03, the predictions for "Weekends" are improved on average 13% when using ratings for "Weekends" instead of using the whole rating set. Similarly, for a distance equal to 0.06, the predictions for "Weekdays" are improved around 11%. Also, larger distance thresholds values result in larger MAE scores, that is, the quality of the recommendations decreases with the user distance threshold.

Finally, we have performed t-tests to see if there are statistically significant differences between the proposed approaches and the time-free model. The results of the tests demonstrate that the probability of the difference being due to chance is less than 0.005, 0.0005 and 0.0005 for the sliding window, damped window and temporal context model, respectively. So clearly, our approaches produce statistically significant recommendations compared to the time-free model.

To summarize, time plays an important role towards improving the quality of the proposed recommendations. The sliding window and the context-based approaches increase the recommendations accuracy. However, a mere decay model

seems to be not adequate. In our current work, we aim at designing a more elaborate aging scheme that considers not only the age of the ratings but also other parameters, such as the recency and popularity of the recommended items and the context under which the ratings were given.

7 Prototype Implementation: *Movie Guide*

To demonstrate the feasibility of our approach, we have developed a research prototype for a movie recommendation application, called tRecs: *A Time-aware Movie Guide* (Fig. 12). The overall system architecture of tRecs is the one depicted in Fig. 4. We maintain information about movies, users and ratings. The movies database schema consists of a single relation with schema: *Movies(mid, title, year, director, genre, language, duration)*. The prototype is implemented in Java and MySQL.

When a user joins the system, he/she registers his/her ratings for computing recommendations and his/her value and attribute preferences for constructing summaries and presenting the results (Fig. 13). Users express their ratings for movies by providing a numerical score between 0.0 and 1.0. Furthermore, users are allowed to define their value-based and attribute-based preferences following either the qualitative or the quantitative preference model. For instance, a user may define that *A. Hitchcock* $\succ$ *M. Scorsese* or may give the score 0.8 to *A. Hitchcock* and the score 0.6 to *M. Scorsese*. Similarly, for the attribute preferences. Clearly, a user can add, delete or modify ratings and preferences at any time and not only at registration time.

Besides user registration, the other part of the application includes recommendations computation, summary construction and presentation. Recommendations computation runs in two modes: time-free and time-aware. In the time-free mode, the temporal aspects of the user ratings are completely ignored. The time-aware mode distinguishes between fresh-based and context-based recommendations. In this mode, the user query is enhanced with some temporal information which is provided by the user as part of his/her query or postulated by the application. In the latter case, the information implicitly associated with the query corresponds to the current temporal characteristics, that is, the context at the time of the submission of the query. The user should also provide the exact scheme that will be used (damped window, sliding window or temporal context).

Presentation summaries are produced with respect to the top-100 time-aware recommendations. User preferences are employed for constructing ordered sets of keywords, in the form of a lattice. Following this ordering, keyword-based summaries, that is, keywords extended with movie titles and production years, are presented to users. As a case study scenario, suppose that a user gave two value-based preferences (*A. Hitchcock* is preferred over *M. Scorsese* and *horror* movies are preferred over *crime* movies) and one attribute-based preference (the *director* of a movie is as important as its *genre*). Suppose also that our user opts to follow the damped window model and would like to know the recommended movies according to his/her previously submitted ratings and preferences. The results of

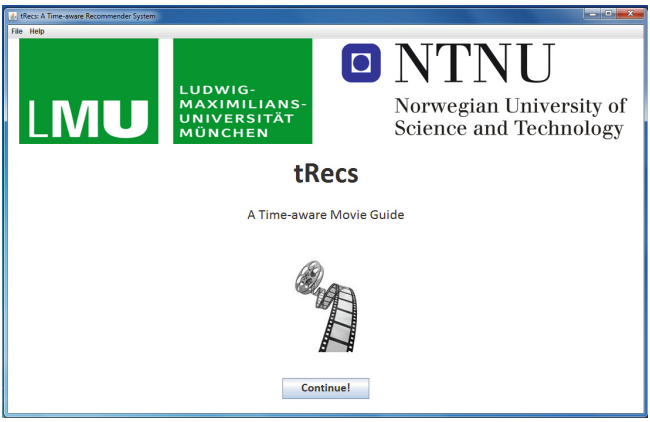


Fig. 12. tRecs: A Time-aware Movie Guide

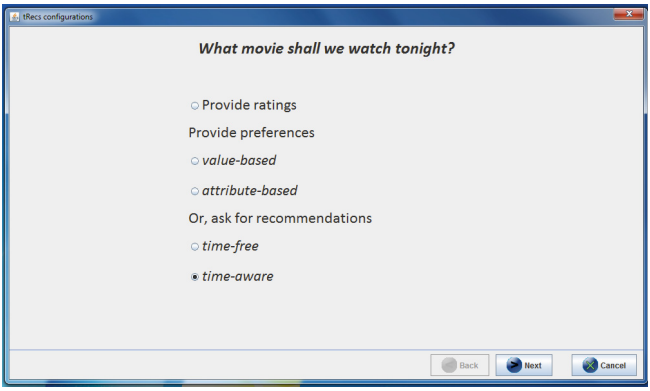


Fig. 13. tRecs configurations

this query example are depicted in Fig. 14. Note that the summary for {*A. Hitchcock, crime*} does not appear in the results, since there are no *crime* movies directed by *A. Hitchcock* in our database instance.

8 Related Work

The research literature on recommendations is extensive. Typically, recommendation approaches are distinguished between: *content-based*, that recommend items similar to those the user previously preferred (e.g., [33,28]), *collaborative filtering*, that recommend items that users with similar preferences liked (e.g., [21,11]) and *hybrid*, that combine content-based and collaborative ones (e.g., [8]). Several extensions have been proposed, such as employing multi-criteria ratings (e.g., [2]) and defining recommendations for groups (e.g., [7,31,30]).

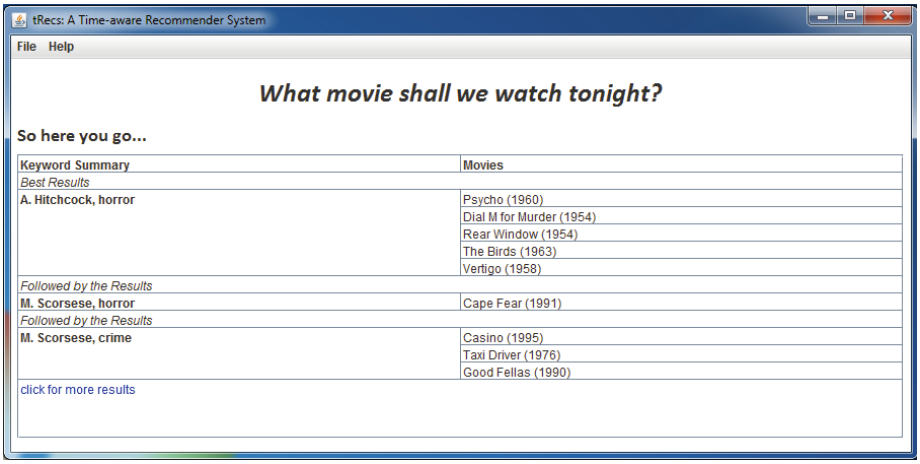


Fig. 14. tRecs: A Time-aware Movie Guide

Recently, there are also approaches focusing on enhancing recommendations with further contextual information (e.g., [3,32]). In these approaches, context is defined as a set of dimensions, or attributes, such as location, companion and time, with hierarchical structure. While a traditional recommendation system considers only two dimensions that correspond to users and items, a context-aware recommendation system considers one additional dimension for each context attribute. In our approach, we focus on a particular case of this model, that is, the three-dimensional recommendations space among users, items and time, since our specific goal is to study how the time effects contribute to the improvement of predictions.

Moreover, there are some approaches which incorporate temporal information to improve recommendations effectiveness. [37] presents a graph-based recommendation system that mixes long-term and short-term user preferences to improve predictions accuracy, while [36] considers how time can be used into matrix factorization models by examining changes in user and society tastes and habits, and items popularity. [15] uses a strategy, similar to our damped window model, that decreases the importance of known ratings as time distance from recommendation time increases. However, the proposed algorithm uses clustering to discriminate between different kinds of items. [10] introduces the idea of micro-profiling, which splits the user preferences into several sets of preferences, each representing the user in a particular temporal context. The predictions are computed using these micro-profiles instead of a single user model. The main focus of this work is on the identification of a meaningful partition of the user preferences using implicit feedback. In our paper, the goal is to examine time from different perspectives. This way, we use a general model for time, considering time either as specific time instances or specific temporal conditions, in order to define a unified time-aware recommendation model.

The temporal aspect of the data has been also studied in different application domains like time series [29] and temporal database queries [14]. Recently the focus has been on huge amounts of data that are collected over time, the so called data streams [4,18]. Due to the theoretically infinite nature of these data, it is impossible to consider them all for answering a query or for a data mining task. So, the rationale is to use the temporal information in order to "reduce" the dataset complexity, e.g., by focusing on a specific period of time instead of the whole stream (e.g., [5]) or by considering the aging of the data (e.g., [12]).

Finally, the general concept of summaries resembles the notion of *tag clouds*. A tag cloud is a visual representation for text data. Tags are usually single words, alphabetically listed and in different font size and color in order to show their importance[1]. Tag clouds have appeared on several Web sites, such as Flickr and del.icio.us, while recently tag cloud drawing has also received attention (e.g., [24]). With regard to summaries for keyword queries, *data clouds* [23] are the most relevant. This work proposes algorithms that try to discover good, not necessarily popular, keywords within the query results. Our approach follows a preference-based technique to locate important keywords. From a different perspective, [16] introduces the notion of *object summary* for summarizing the data in a relational database about a particular *data subject*, or keyword. An object summary is a tree with a tuple containing the keyword as the root node and its neighboring tuples containing additional information as child nodes. [17] extends this work by presenting a partial object summary of size l, composed of only l representative tuples.

9 Conclusions

In this paper, we study different semantics to exploit the time information associated with user ratings in order to improve the accuracy of recommendations. We consider various types of time effects, and thus, propose different time-aware recommendation models. Fresh-based recommendations care mainly for recent and novel ratings, while context-based recommendations are computed with respect to ratings with temporal context equal to the query context. To help users receive a broader view of the recommended items, we add some structure to the presentation of the results. In particular, we rank the recommended items based on user preferences and organize equally important items through summaries. Finally, we evaluate our approach using a real dataset of movie ratings and demonstrate its feasibility through a prototype implementation of a movie guide application.

There are several directions for future work. We envision to extend our framework so as to support a novel mode of interaction between users and recommendation systems; our goal is to exploit the whole rating history to produce valued recommendations and, at the same time, use the fresh ratings to assist users in database exploration.

[1] `en.wikipedia.org/wiki/Tag_cloud`

Acknowledgments. The work of the second author is supported by the project "IdeaGarden" funded by the Seventh Framework Programme under grand n^o 318552.

References

1. Movielens data sets, `http://www.grouplens.org/node/12` (visited on November 2011)
2. Adomavicius, G., Kwon, Y.: New recommendation techniques for multicriteria rating systems. IEEE Intelligent Systems 22(3), 48–55 (2007)
3. Adomavicius, G., Sankaranarayanan, R., Sen, S., Tuzhilin, A.: Incorporating contextual information in recommender systems using a multidimensional approach. ACM Trans. Inf. Syst. 23(1), 103–145 (2005)
4. Aggarwal, C.C. (ed.): Data Streams – Models and Algorithms. Springer (2007)
5. Aggarwal, C.C., Han, J., Wang, J., Yu, P.: A framework for clustering evolving data streams. In: VLDB (2003)
6. Agrawal, R., Wimmers, E.L.: A framework for expressing and combining preferences. In: SIGMOD Conference, pp. 297–306 (2000)
7. Amer-Yahia, S., Roy, S.B., Chawla, A., Das, G., Yu, C.: Group recommendation: Semantics and efficiency. PVLDB 2(1), 754–765 (2009)
8. Balabanovic, M., Shoham, Y.: Content-based, collaborative recommendation. Commun. ACM 40(3), 66–72 (1997)
9. Balog, K., Bogers, T., Azzopardi, L., de Rijke, M., van den Bosch, A.: Broad expertise retrieval in sparse data environments. In: SIGIR, pp. 551–558 (2007)
10. Baltrunas, L., Amatriain, X.: Towards time-dependant recommendation based on implicit feedback. In: CARS, pp. 1–5 (2009)
11. Breese, J.S., Heckerman, D., Kadie, C.M.: Empirical analysis of predictive algorithms for collaborative filtering. In: UAI, pp. 43–52 (1998)
12. Cao, F., Ester, M., Qian, W., Zhou, A.: Density-based clustering over an evolving data stream with noise. In: SDM 2006 (2006)
13. Chomicki, J.: Preference formulas in relational queries. ACM Trans. Database Syst. 28(4), 427–466 (2003)
14. Dayal, U., Wuu, G.T.J.: A uniform approach to processing temporal queries. In: VLDB, pp. 407–418 (1992)
15. Ding, Y., Li, X.: Time weight collaborative filtering. In: CIKM, pp. 485–492 (2005)
16. Fakas, G.J.: A novel keyword search paradigm in relational databases: Object summaries. Data Knowl. Eng. 70(2), 208–229 (2011)
17. Fakas, G.J., Cai, Z., Mamoulis, N.: Size-l object summaries for relational keyword search. PVLDB 5(3), 229–240 (2011)
18. Gama, J.: Knowledge Discovery from Data Streams. CRC Press (2010)
19. Georgiadis, P., Kapantaidakis, I., Christophides, V., Nguer, E.M., Spyratos, N.: Efficient rewriting algorithms for preference queries. In: ICDE, pp. 1101–1110 (2008)
20. Kießling, W.: Foundations of preferences in database systems. In: VLDB, pp. 311–322 (2002)
21. Konstan, J.A., Miller, B.N., Maltz, D., Herlocker, J.L., Gordon, L.R., Riedl, J.: Grouplens: Applying collaborative filtering to usenet news. Commun. ACM 40(3), 77–87 (1997)
22. Koutrika, G., Ioannidis, Y.E.: Personalizing queries based on networks of composite preferences. ACM Trans. Database Syst. 35(2) (2010)

23. Koutrika, G., Zadeh, Z.M., Garcia-Molina, H.: Data clouds: summarizing keyword search results over structured data. In: EDBT, pp. 391–402 (2009)
24. Kuo, B.Y.-L., Hentrich, T., Good, B.M., Wilkinson, M.D.: Tag clouds for summarizing web search results. In: WWW, pp. 1203–1204 (2007)
25. Melville, P., Sindhwani, V.: Recommender systems. In: Encyclopedia of Machine Learning, pp. 829–838 (2010)
26. Miele, A., Quintarelli, E., Tanca, L.: A methodology for preference-based personalization of contextual data. In: EDBT, pp. 287–298 (2009)
27. Mobasher, B., Cooley, R., Srivastava, J.: Automatic personalization based on web usage mining. Commun. ACM, 142–151 (2000)
28. Mooney, R.J., Roy, L.: Content-based book recommending using learning for text categorization. In: ACM DL, pp. 195–204 (2000)
29. Nong, Y.: The Handbook of Data Mining. Lawrence Erlbaum Associates, Mahwah (2003)
30. Ntoutsi, E., Stefanidis, K., Nørvåg, K., Kriegel, H.-P.: Fast group recommendations by applying user clustering. In: Atzeni, P., Cheung, D., Ram, S. (eds.) ER 2012 Main Conference 2012. LNCS, vol. 7532, pp. 126–140. Springer, Heidelberg (2012)
31. O'Connor, M., Cosley, D., Konstan, J.A., Riedl, J.: Polylens: A recommender system for groups of user. In: ECSCW, pp. 199–218 (2001)
32. Palmisano, C., Tuzhilin, A., Gorgoglione, M.: Using context to improve predictive modeling of customers in personalization applications. IEEE Trans. Knowl. Data Eng. 20(11), 1535–1549 (2008)
33. Pazzani, M.J., Billsus, D.: Learning and revising user profiles: The identification of interesting web sites. Machine Learning 27(3), 313–331 (1997)
34. Ricci, F., Rokach, L., Shapira, B., Kantor, P.B.: Recommender Systems Handbook. Springer, New York, Inc. (2010)
35. Stefanidis, K., Pitoura, E., Vassiliadis, P.: Managing contextual preferences. Inf. Syst. 36(8), 1158–1180 (2011)
36. Xiang, L., Yang, Q.: Time-dependent models in collaborative filtering based recommender system. In: Web Intelligence, pp. 450–457 (2009)
37. Xiang, L., Yuan, Q., Zhao, S., Chen, L., Zhang, X., Yang, Q., Sun, J.: Temporal recommendation on graphs via long- and short-term preference fusion. In: KDD, pp. 723–732 (2010)

Incremental Mining of Top-k Maximal Influential Paths in Network Data

Enliang Xu, Wynne Hsu, Mong Li Lee, and Dhaval Patel

School of Computing, National University of Singapore
{xuenliang,whsu,leeml,dhaval}@comp.nus.edu.sg

Abstract. Information diffusion refers to the spread of abstract ideas and concepts, technical information, and actual practices within a social system, where the spread denotes flow or movement from a source to an adopter, typically via communication and influence. Discovering influence relations among users has important applications in viral marketing, personalized recommendations and feed ranking in social networks. Existing works on information diffusion analysis have focused on discovering "influential users" and "who influences whom" relationships using data obtained from social networks. However, they do not consider the continuity of influence among users. In this paper, we develop a method for inferring top-*k* maximal influential paths which can capture the continuity of influence. We define a generative influence propagation model based on the Independent Cascade Model and Linear Threshold Model, which mathematically models the spread of certain information through a network. We formalize the top-*k* maximal influential path inference problem and develop an efficient algorithm, called TIP, to infer the top-*k* maximal influential paths. TIP makes use of the properties of top-*k* maximal influential paths to dynamically increase the support and prune the projected databases. As databases evolve over time, we also develop an incremental mining algorithm IncTIP to maintain top-*k* maximal influential paths. We evaluate the proposed algorithms on both synthetic and real-world datasets. The experimental results demonstrate the effectiveness and efficiency of both TIP and IncTIP.

1 Introduction

With the prevalence of online social media such as Facebook, Twitter and YouTube, information diffusion analysis has attracted great research interests recently. Information diffusion refers to the spread of abstract ideas and concepts, technical information, and actual practices within a social system, where the spread denotes flow or movement from a source to an adopter, typically via communication and influence [1]. Discovering influence relations among users has important applications in viral marketing, personalized recommendations and feed ranking in social networks. Existing works on information diffusion analysis have focused on discovering "influential users" [10,11,12,18,5,4] and "who influences whom" relationships [7,17] using data obtained from social networks. However, they either assume the existence of a social graph with edges labeled with influence probabilities or do not consider the continuity of influence among users. The notion of "continuity of influence" is inspired by the fact that, after adopting an

A. Hameurlain et al. (Eds.): TLDKS X, LNCS 8220, pp. 173–199, 2013.

action, users influence other users who have not performed similar action before, thus triggering a cascade of influence. Based on this observation, we introduced the concept of "influential path" that captures the continuity of influence [21].

Figure 1 shows the top-5 influential paths obtained from the MemeTracker dataset [13]. Each node in the network is a news website and a directed edge from node a to node b indicates that information has propagated from a to b. For example, {`us.rd.yahoo.com` $\rightarrow$ `seattletimes.nwsource.com` $\rightarrow$ `blog.beliefnet.com`} is an example of top-1 influential path. Our quick analysis of this pattern revealed that a new piece of information gets propagated from `us.rd.yahoo.com` to `seattletimes.nwsource.com` to `blog.beliefnet.com`. This information can be utilized in automated news feeding application, where we inform website `seattletimes.nwsource.com` about newly published information by `us.rd.yahoo.com` and so on. Apart from this news feeding application, top-k influential paths are also useful for finding critical nodes which should have mirror sites. For example, website `us.rd.yahoo.com` is present in 4 of the top-5 influential paths, then any disruptions to this node may lead to news blackout.

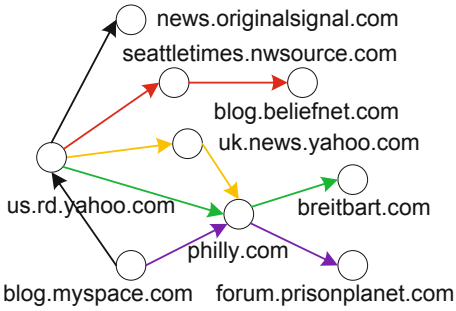


Fig. 1. Top-5 influential paths in MemeTracker Dataset

The problem is further complicated by the fact that users of the aforementioned online social media are active and regularly upload new information to the social media. For example, news websites regularly publish important information in MemeTracker dataset. On average, 20,000 news articles are produced per hour from August 2008 to January 2009 in the MemeTracker dataset (see Figure 2). Such updates may introduce new patterns or invalidate some existing patterns. Recomputing top-k maximal influential paths for each update is very inefficient.

In this paper, we describe a method, called TIP to infer the top-k maximal influential paths which can truly capture the dynamics of information diffusion among users in the social network [21]. Given a log of propagation observations of some information over a hidden network, our goal is to infer the top-k maximal influential paths that best explain these observations. We define a generative influence propagation model based on the Independent Cascade Model and the Linear Threshold Model, which mathematically models the spread of certain information through a network. By utilizing the properties

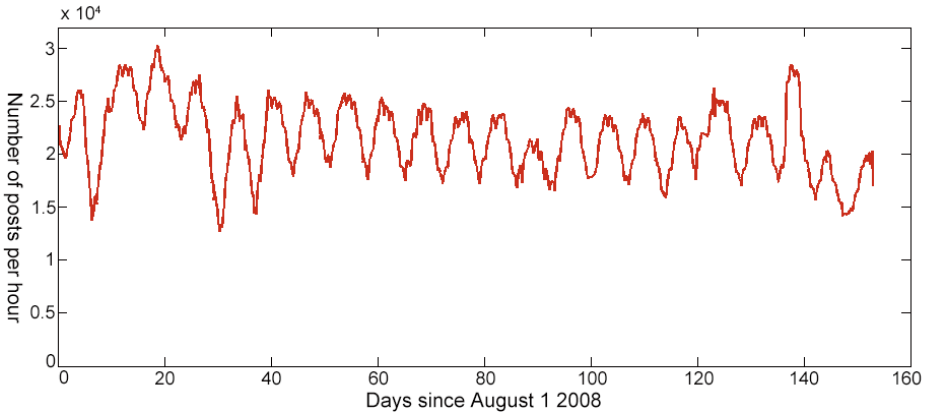


Fig. 2. News updates in Memetracker dataset

of top-k maximal influential paths, TIP can dynamically increase the support and prune the projected databases.

We then extend TIP to allow for incremental mining. The extended algorithm, named IncTIP, leverages on the computation performed in previous stages to maintain the set of top-k maximal influential paths efficiently. We evaluate the proposed algorithms on both synthetic and real-world datasets. The experimental results demonstrate the effectiveness and efficiency of both TIP and IncTIP.

2 Influence Propagation Model

An influence network aims to capture the propagation of influence among a set of entities based on a list of observations. We model the network using a directed graph $G = (V, E)$ where V and E are the sets of nodes and edges respectively.

A node u in V denotes an entity and can be *active* or *inactive*. It is considered *active* if it has been influenced. Nodes can switch from being inactive to active, but not vice versa. When a node u gets influenced, it in turns may influence each of its currently inactive neighbors v with some small probability. Node u can only influence its neighbor v if their time difference is within some time threshold τ.

Each directed edge $(u,v) \in E$ has a weight $weight(u,v) \in [0,1]$ denoting the likelihood of node v being influenced by node u. Suppose t_u and t_v are the times at which nodes u and v get influenced respectively. Then $weight(u,v) = 0$ if $t_v \leq t_u$, i.e., nodes cannot be influenced by nodes from the future time points. Otherwise, $weight(u,v) = e^{-\frac{t_v - t_u}{\alpha}}$ where α is radius of influence.

We associate each node u with an influence measure which is computed from the weights of the edges connecting u to its active neighboring nodes as follows:

$$influence(u,S) = 1 - \prod_{w \in S} (1 - weight(w,u)) \tag{1}$$

where S is the set of active neighbors of u.

One immediate concern is the cost of updating $influence(u,S)$ when the status of nodes change. Since the node status changes frequently, this update cost can be computationally expensive. We derive an expression that allows $influence(u,S)$ to be updated incrementally.

Suppose a new neighboring node w of u becomes active. Then

$$\begin{aligned} influence(u,S\cup\{w\}) &= 1-(1-weight(w,u))*\prod_{u'\in S}(1-weight(u',u)) \\ &= 1-(1-weight(w,u))*(1-influence(u,S)) \\ &= influence(u,S)+(1-influence(u,S))*weight(w,u) \quad (2) \end{aligned}$$

We observe that the influence measure $influence(u,S)$ is both monotonic and sub-modular.

A function $f(.)$ is *monotonic* if $f(S)\leq f(S')$, for $S\subseteq S'$. From Equation 2, we have

$$influence(u,S\cup\{w\})-influence(u,S)=(1-influence(u,S))*weight(w,u)\geq 0$$

A function $f(.)$ is *submodular* if $f(S\cup\{w\})-f(S)\geq f(S'\cup\{w\})-f(S')$, for $S\subseteq S'$. This means that adding a node w to S increases the score more than adding w to S' when $S\subseteq S'$. We show that $influence(u,S)$ is sub-modular as follows:

$$\begin{aligned} &influence(u,S\cup\{w\})-influence(u,S)-(influence(u,S'\cup\{w\})-influence(u,S')) \\ &\quad=(1-influence(u,S))*weight(w,u)-(1-influence(u,S'))*weight(w,u) \\ &\quad=(influence(u,S')-influence(u,S))*weight(w,u) \quad (3) \end{aligned}$$

By monotonicity, $influence(u,S')\geq influence(u,S)$. Hence,

$$(influence(u,S')-influence(u,S))*weight(w,u)\geq 0$$

Definition 1. *An observation* $o = <(u_1,t_1),(u_2,t_2),\cdots,(u_n,t_n)>$ *is a sequence of tuples* (u_i,t_i) *where* t_i *is the time when node* u_i *becomes active, and* $\forall\ i<j,\ t_i<t_j$. *Further,* $u_i\neq u_j\ \forall\ i\neq j$. *The length of observation o, denoted as* $|o|$, *is the number of* (u_i,t_i) *tuples in o.*

Definition 2. *An influential path is a sequence of nodes, denoted as* $p = <v_1\rightarrow v_2\rightarrow\cdots\rightarrow v_n>$, *such that weight*$(v_i,v_{i+1})$ *is larger than some user defined threshold for all* i, $1\leq i\leq n-1$. *The length of p is given by* $|p| = n-1$.

Definition 3. *An observation o supports an influential path p if*

- $\forall v\in p,\ v\in\{\ u_i\mid(u_i,t_i)\in o\}$, *and*
- *if* u_i *and* u_j *are nodes in o that correspond to* $v_{i'}$ *and* $v_{i'+1}$, *then* $0<t_j-t_i<\tau$, $1\leq i'\leq n-1$.

Let D be an observation database, which consists of a set of observations. The *support* of an influential path p, denoted as $support(p)$, is the fraction of observations in D that support p.

The *score* of a path $p = < v_1 \rightarrow v_2 \rightarrow \cdots \rightarrow v_n >$ w.r.t. an observation o is defined as

$$score(p,o) = \log(influence(v_1,S) \prod_{1 \leq i \leq n-1} weight(v_i, v_{i+1})) - \log \varepsilon, \quad (4)$$

where $\varepsilon \in [0,1]$ is some small value and S is the set of active neighbors of v_1 w.r.t. o.

Let S_p be the set of observations in D that support influential path p. The *total score* of p, denoted as $total\_score(p)$, is defined by

$$total\_score(p) = \sum_{o \in S_p} score(p,o). \quad (5)$$

An influential path $p = < v_1 \rightarrow v_2 \rightarrow \cdots \rightarrow v_m >$ is a *sub-path* of another influential path $p' = < v'_1 \rightarrow v'_2 \rightarrow \cdots \rightarrow v'_n >$, denoted as $p \sqsubseteq p'$, if and only if $\exists\ i_1, i_2, \cdots, i_m$, such that $1 \leq i_1 < i_2 < \cdots < i_m \leq n$, and $v_1 = v'_{i_1}$, $v_2 = v'_{i_2}$, $\cdots$, and $v_m = v'_{i_m}$. We also call p' a *super-path* of p.

An influential path p is **maximal** if there exists no influential path p' such that $p \sqsubseteq p'$ and support(p) = support(p').

Definition 4. *An influential path p is a top-k maximal influential path if p is maximal and there exist no more than* $(k-1)$ *maximal influential paths whose total score is greater than that of p.*

The following theorem states the relation between the support and total score of two maximal influential paths. This theorem is utilized by our proposed algorithm in Section 3 to effectively prune off the search space.

Theorem 1. *For any two maximal influential paths p and* p'*, if* $support(p) > support(p')$ *and* $\varepsilon < e^{-|D|(|o|+1)\tau}$*, then* $total\_score(p) > total\_score(p')$ *where o is an observation with maximum length in database D.*

Proof. Let p be a maximal influential path with support s and length $|p|$. We can calculate the total score of path p as

$$\begin{aligned}
total_score(p) &= \sum_{o \in S_p} score(p,o) \\
&> (\log e^{-\tau} + |p| * \log e^{-\tau} - \log \varepsilon) * s \\
&= -s\tau - s|p|\tau - s * \log \varepsilon \\
&= -s\tau - s|p|\tau - s * \log \varepsilon + \log \varepsilon - \log \varepsilon \\
&= (-\log \varepsilon) * (s-1) + (-s\tau - s|p|\tau - \log \varepsilon) \\
&> (-\log \varepsilon) * (s-1) + (-s\tau - s|p|\tau - \log e^{-|D|(|o|+1)\tau}) \\
&= (-\log \varepsilon) * (s-1) + (|D|(|o|+1) - s(|p|+1))\tau \\
&> (-\log \varepsilon) * (s-1)
\end{aligned}$$

Since $(|D|(|o|+1) - s(|p|+1)) \geq 0$, we have $(\log e^{-\tau} + |p| * \log e^{-\tau} - \log \varepsilon) * s > (-\log \varepsilon) * (s-1)$. Note that $(\log e^{-\tau} + |p| * \log e^{-\tau} - \log \varepsilon) * s$ is the lower bound for the $total\_score$ of any maximal influential path with support s, and $(-\log \varepsilon) * (s-1)$

is the upper bound for the *total_score* of any maximal influential path with support $(s-1)$. Further, the value of *total_score* decreases with the length of a path. Hence, $(\log e^{-\tau} + |p| * \log e^{-\tau} - \log \varepsilon) * s > (-\log \varepsilon) * (s-1)$ implies that the *total_score* of any maximal influential path with support s is greater than all the maximal influential paths whose support is less than s. □

3 The TIP Algorithm

In this section, we first describe the TIP algorithm that mines the top-k maximal influential paths without having to specify a minimum support threshold [21]. TIP is a prefix-based influential path mining method. It extends the classical projection-based pattern growth method [14] with time constraint. Instead of projecting observation databases by considering all possible occurrences of prefixes, TIP examines the frequent prefix sub-paths and projects only the corresponding valid observations which satisfy the time constraint into the projected databases. The influential paths are then extended by exploring the valid frequent nodes in the projected databases.

Given an influential path $p = < v_1 \rightarrow v_2 \rightarrow \cdots \rightarrow v_n >$ and a node α, we can extend p by α if the last node of p, i.e. v_n, can $influence$ α, that is, the time difference between t_{v_n} and t_α is within the time threshold τ. We denote the extension as $p \rightarrow \alpha = < v_1 \rightarrow v_2 \rightarrow \cdots \rightarrow v_n \rightarrow \alpha >$.

Let $p' = p \rightarrow \alpha$ be an extension of p. we say p is a *prefix* of p' and α is a *suffix* of p'. For example, in our sample observation database D as shown in Table 1, $< a \rightarrow d \rightarrow g >$ is a prefix of path $< a \rightarrow d \rightarrow g \rightarrow i >$ and $< i >$ is its suffix.

Let S_p be the set of observations that support influential path p. Suppose each $o \in S_p$ is of the form $< (u_1,t_1), (u_2,t_2), \cdots, (u_a,t_a), (u_{a+1},t_{a+1}), \cdots, (u_b,t_b) >$. Then we define the *p-projected* database as $D_p = \{ < (u_{a+1},t_{a+1}), \cdots, (u_b,t_b) > \}$ if the last node $v_n \in p$ corresponds to $u_a \in o$ and the time difference $t_{a+1} - t_a$ is less than τ.

Table 1. A Sample Observation Database D

ID	Observation
o_1	<(a,1) (d,5) (g,10) (i,16)>
o_2	<(c,8) (e,15) (f,20)>
o_3	<(c,4) (d,10) (g,16) (i,20)>
o_4	<(c,3) (e,12) (i,36)>
o_5	<(c,5) (e,9) (h,20) (i,24)>

Table 2. Frequent nodes in D

Node	Count
c	4
i	4
e	3
d	2
g	2
a	1
f	1
h	1

Consider the sample observation database in Table 1. Let time threshold $\tau = 20$. The projected database for path $< c \rightarrow e >$ is $D_{<c \rightarrow e>} = \{< (f,20) >, < (h,20), (i,24) >\}$. Note that for observation o_4, the time stamp of e is 12, while the next time stamp is 36. Since the time difference is 24 which is more than τ, node e cannot influence node i, and hence $< (i,36) >$ is not included in the projected database.

Algorithm 1. TIPMiner(D, k, τ)

Require: global variable *PathSet*
Require: observation database D, an integer k and time threshold τ
Ensure: Top-k maximal influential path set *PathSet*
1: $V \leftarrow$ nodes in D
2: Initialize $min\_sup = 1$
3: Initialize $PathSet = \emptyset$
4: Let *root* be the root node
5: **for each** node $v \in V$ **do**
6: Create child node v of *root* and record support count and IDs of the supporting observations of v
7: Update *PathSet* by calling TIP(v, $D_{<v>}$, k, min_sup, τ, *PathSet*)
8: **end for**
9: return *PathSet*

Having defined the concept of path-projected databases, we next describe the framework TIPMiner for mining the top-k maximal influential paths from a given observation database D. Algorithm 1 gives the details. It first finds all the nodes in D and sorts them in decreasing order of their support values. A global variable *PathSet* is used to keep track of the set of top-k maximal influential paths. This global variable is updated by calling Algorithm TIP (see Algorithm 2) for each node.

Algorithm TIP finds the top-k maximal influential paths by constructing projected databases. Inputs to TIP algorithm are an influential path p, the p-projected database D_p, the number of maximal influential paths k, minimum support threshold min_sup, time threshold τ, and *PathSet*. The outputs are the set of top-k maximal influential paths *PathSet*.

Given an influential path p, TIP algorithm attempts to extend p by first obtaining the p-projected database D_p. Initially, the path consists of only one node. Given a path p, we first check if this path is promising (lines 1-3). A path is promising if its support is no less than the minimum support threshold. We calculate the $total\_score$ of path p (lines 4-5). Line 6 checks whether there exists an influential path $p' \in PathSet$ such that p is a sub-path or super-path of p'. If p' exists, we perform maximal influential path verification (lines 8-15). If p' is a sub-path of p, then we replace p' by p in the *PathSet* since p is now the maximal influential path (lines 12-14). However, if p' is a super-path of p, then p is not a maximal influential path and can be discarded (lines 9-11).

If p' does not exist and *PathSet* contains less than k maximal influential paths, then we add p to the *PathSet* (lines 17-18). Otherwise, if *PathSet* already contains k maximal influential paths, we check the $total\_score$ of p. If the $total\_score$ of p is larger than any of the k maximal influential paths in *PathSet*, we replace the path with the smallest $total\_score$ by p (lines 19-24). By Theorem 1, we raise min_sup to the support of the path whose $total\_score$ is the minimum in *PathSet* (lines 26-29). This allows us to prune off unpromising paths.

Next, the algorithm attempts to extend p by finding all the frequent nodes $\alpha \in D_p$ such that we can extend p to $p \rightarrow \alpha$ (lines 30-41). We scan the p-projected database D_p to find every frequent node α, such that path p can be extended to $p \rightarrow \alpha$, and insert α into a priority queue Q (lines 31-36). We recursively call TIP algorithm to extend

Algorithm 2. TIP(p, D_p, k, min_sup, τ, $PathSet$)

Require: global variable $PathSet$, min_sup
Require: a path p, D_p, an integer k and time threshold τ
Ensure: Top-k maximal influential path set $PathSet$
if $support(p) < min\_sup$ **then**
  return
end if
let S_p be the set of observations that support p
calculate $total\_score(p) = \sum_{o \in S_p} score(p, o)$
check whether a discovered influential path $p' \in PathSet$ exists, s.t. either $p \sqsubseteq p'$ or $p' \sqsubseteq p$, and $support(p) = support(p')$
if such super-path or sub-path exists **then**
  for each $p' \in PathSet$ such that $support(p') = support(p)$ **do**
    if $p \sqsubseteq p'$ **then**
      return
    end if
    if $p' \sqsubseteq p$ **then**
      replace p' with p
    end if
  end for
else
  if $|PathSet| < k$ **then**
    $PathSet = PathSet \cup \{p\}$
  else
    let path $q \in PathSet$ such that $\nexists\, q' \in PathSet, total\_score(q') < total\_score(q)$
    if $total\_score(p) > total\_score(q)$ **then**
      replace q with p
    end if
  end if
end if
if $|PathSet| = k$ **then**
  let path $q \in PathSet$ such that $\nexists\, q' \in PathSet, total\_score(q') < total\_score(q)$
  $min\_sup = support(q)$
end if
$Q \leftarrow$ empty priority queue
compute the frequency of each node in D_p
for each frequent node α **do**
  if p can be extended to $p \rightarrow \alpha$ **then**
    Q.insert(α)
  end if
end for
while !Q.isEmpty() **do**
  $\alpha = Q$.pop()
  create child node α of the last node of p and record support count and IDs of the supporting observations of α
  Call TIP($p \rightarrow \alpha$, $D_{p \rightarrow \alpha}$, k, min_sup, τ, $PathSet$)
end while
return

another path using the next frequent node in Q (lines 37-41). The algorithm terminates when Q is empty.

Table 3. $< c >$-projected database $D_{<c>}$

ID	Observation
o_2	<(e,15) (f,20)>
o_3	<(d,10) (g,16) (i,20)>
o_4	<(e,12) (i,36)>
o_5	<(e,9) (h,20) (i,24)>

Table 4. Frequent nodes in $D_{<c>}$

Node	Count
e	3
i	2
d	1
f	1
g	1
h	1

Let us now use the example in Table 1 to illustrate the TIP algorithm. The entity with the highest support value is c (see Table 2). We obtain the projected database $D_{<c>}$ as shown in Table 3. The frequent nodes with their support values are shown in Table 4. We insert these nodes into the priority queue Q and recursively call TIP to extend $< c >$. Since node e has support 3 in Q, we extend $< c >$ to $< c \rightarrow e >$.

Conceptually, the TIP algorithm is constructing a prefix search tree where node in the tree corresponds to an influential path starting from the root to the node and its support is shown next to the node as shown in Figure 3. The number along each edge denotes the *total_score* of the path from the root to the end node of the edge. We assume that the time threshold $\tau = 20$ and $\varepsilon = e^{-64}$. We observe that $< c \rightarrow e >$ are supported by three observations o_2, o_4 and o_5 in Table 1. The scores with respect to these observations are as follows:

$$\begin{aligned} score(p, o_2) &= \log(influence(c, S) * weight(c, e)) - \log \varepsilon \\ &= \log e^{-\frac{15-8}{1.0}} - \log e^{-64} \\ &= 57 \end{aligned}$$

Similarly, we have $score(p, o_4) = 55$ and $score(p, o_5) = 60$. Thus the *total_score* of the influential path $p = < c \rightarrow e >$ is $total\_score(p) = 57 + 55 + 60 = 172$. In the same manner, we build $< c \rightarrow e >$-projected database and extend $< c \rightarrow e >$ to $< c \rightarrow e \rightarrow f >$.

Suppose we wish to find the top-2 maximal influential paths. After obtaining the paths $< c \rightarrow e >$ and $< c \rightarrow i >$, the *min_sup* is raised to 2. This implies that all the branches rooted at node a are pruned as their support values are less than 2. Similarly, branches rooted at node e are also pruned as they have already been traversed previously from node c. The bold lines in Figure 3 show the explored paths.

To further improve the efficiency of TIP algorithm, we propose two optimization strategies.

Early Termination by Equivalence. Early termination by equivalence is a search space reduction technique developed in CloSpan [15]. Let $N(D)$ represent the total

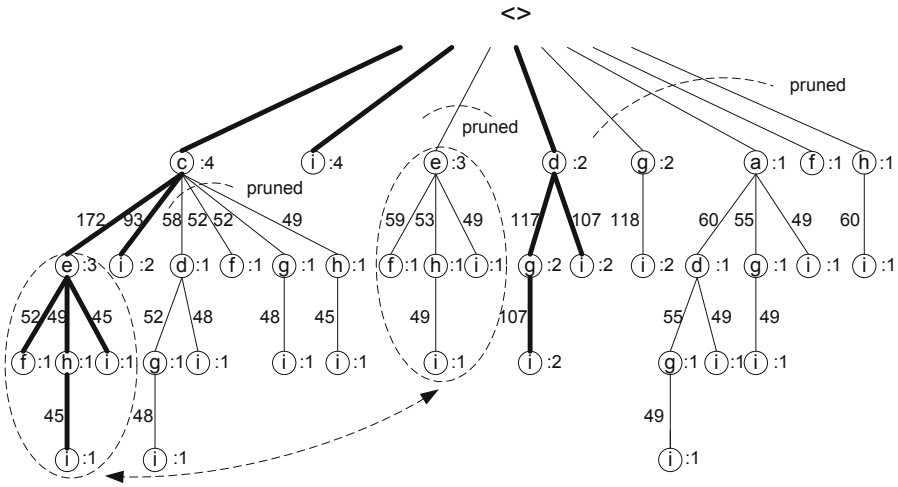


Fig. 3. Prefix search tree for sample database

number of nodes in D. The property of early termination by equivalence shows if two influential paths $p \sqsubseteq p'$ and $N(D_p) = N(D_{p'})$, then $\forall\ \gamma$, $support(p \rightarrow \gamma) = support(p' \rightarrow \gamma)$. It means the descendants of p in the prefix search tree cannot be maximal. Furthermore, the descendants of p and p' are exactly the same. We can utilize this property to quickly prune the search space of p.

Pseudo Projection. As with traditional projection-based mining method, the major cost of TIP is the construction of projected databases. To reduce the cost of projection, we apply the pseudo-projection technique [14]. Instead of constructing a *physical projection* by collecting all the postfixes, we use pointers referring to the observations in the database as a *pseudo projection*. Every projection consists of two pieces of information: $pointer$ to the observation in database and $offset$ of the postfix in the observation. This allows us to avoid physically copying postfixes: only pointers to the projected point are saved for each observation. Thus, it is efficient in terms of both running time and space.

4 Incremental Mining

One challenge in finding the top-k maximal influential paths in social networks is that most users are active and updates tend to be frequent and voluminous. In general, there are three kinds of updates: (1) new observation arrives. This corresponds to an INSERT operation. (2) new follow-up action is observed later. This corresponds to an APPEND operation. (3) an existing observation is no longer valid and should be removed. This corresponds to a DELETE operation. We can consider APPEND as deleting an existing observation and inserting a new one. For example, if we wish to append the tuple $< (g,26) >$ to observation o_2 in Table 1, we first delete o_2 and insert the observation o_2': $< (c,8)(e,15)(f,20)(g,26) >$ into D.

Invoking TIP for each update is infeasible. In this section, we describe an incremental mining algorithm to mine top-*k* maximal influential paths. The main idea in incremental mining is to leverage on the computations done previously. In order to do this, we need to store additional information for each node, namely the support count for each of its extended child and the IDs of the supporting observations. Figure 4 shows the additional information we keep for *root* and node *c*, *e* in the explored paths of the prefix tree.

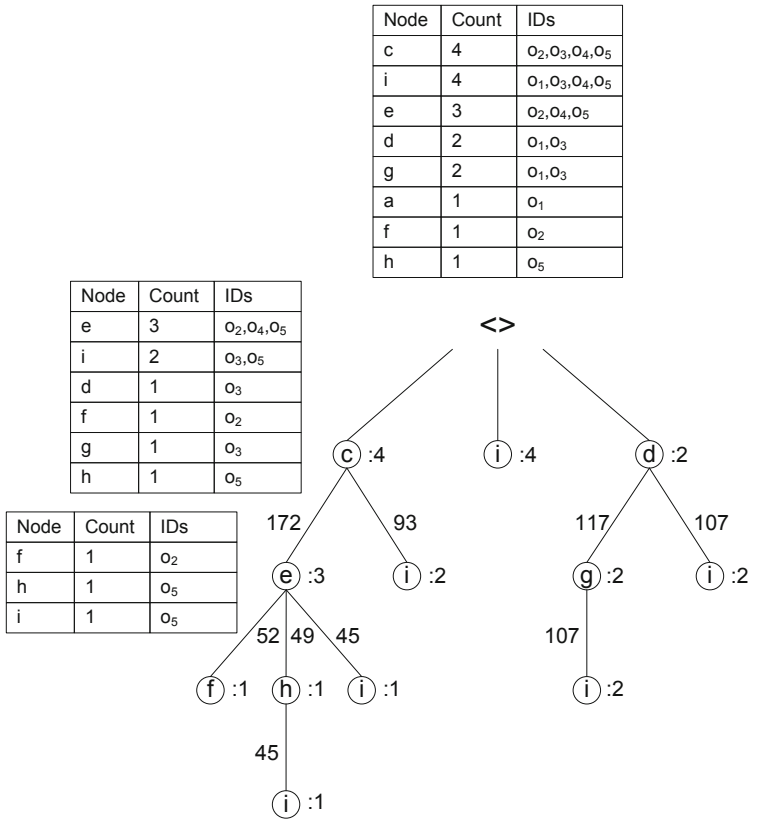


Fig. 4. Prefix tree with additional information for *root* and node *c*, *e*

The IncTIP algorithm for incremental mining of top-*k* maximal influential paths is given in Algorithm 3. The inputs are database *D*, set of updates *U*, an integer *k*, top-*k* maximal influential path set *PathSet* and the corresponding final *min_sup*, and time threshold τ. The output is the set of top-*k* maximal influential paths *PathSet*. For each update, we first check whether it is INSERT or DELETE (line 2). If the update is INSERT an observation *o*, then for each node *v* in *o*, we scan additional information table of *root* and check whether *v* is frequent or not (line 5). If it is frequent, we update the *PathSet* by calling the INSERT algorithm (lines 6-7). Otherwise, we call the TIP algorithm (lines 8-12). If the update is DELETE an observation *o*, we update the *PathSet* by

calling the DELETE algorithm (lines 15-17). The global variable *PathSet*, which keeps track of the set of top-k maximal influential paths, is updated by calling the appropriate algorithms. Algorithm 4 and 5 gives the details of INSERT and DELETE respectively. We will illustrate them in detail in the following subsections with our running example.

Algorithm 3. IncTIP(D, U, k, *min_sup*, τ, *PathSet*)

Require: global variable *PathSet*, *min_sup*
Require: database D, set of updates U, an integer k and time threshold τ
Ensure: Top-k maximal influential path set *PathSet*
1: let *root* be the root node
2: **for each** update in U **do**
3: **if** INSERT o **then**
4: **for each** node $v \in o$ **do**
5: scan additional information table of *root*, check whether v is frequent or not
6: **if** v is frequent **then**
7: Update *PathSet* by calling INSERT(D, v, o, k, *min_sup*, τ, *PathSet*)
8: **else**
9: let I be the set of observations in $D \cup \{o\}$ that support v
10: let $I_{<v>}$ be v-projected database
11: Update *PathSet* by calling TIP(v, $I_{<v>}$, k, *min_sup*, τ, *PathSet*)
12: **end if**
13: **end for**
14: **else**
15: **if** DELETE o **then**
16: Update *PathSet* by calling DELETE(D, *root*, o, k, *min_sup*, τ, *PathSet*)
17: **end if**
18: **end if**
19: **end for**
20: return *PathSet*

4.1 Insert Observation

Suppose we insert a new observation o_6: $< (a,2)(d,7)(i,13) >$ into the sample observation database D in Table 1. The new observation database D' after insertion is shown in Table 5.

Recall that in our previous running example for the TIP algorithm, we find top-2 maximal influential paths and the *min_sup* is finally raised to 2. So all the branches rooted at node a are pruned as their support values are less than 2 (see Figure 3). However, after inserting observation o_6, the support of node a becomes 2, implying that we should mine influential paths starting at node a. Based on the additional information for the root node as shown in Table 6, we know that in the original database observation o_1 supports node a. So observations that support node a are observation o_1 and the inserted observation o_6.

For node a, we call the TIP algorithm (Algorithm 2). We obtain $< a >$-projected database $I_{<a>}$ as shown in Table 7. The frequent nodes with their support values are shown in Table 8. We insert these nodes into the priority queue Q and recursively call

Table 5. New database D' after insertion

ID	Observation
o_1	<(a,1) (d,5) (g,10) (i,16)>
o_2	<(c,8) (e,15) (f,20)>
o_3	<(c,4) (d,10) (g,16) (i,20)>
o_4	<(c,3) (e,12) (i,36)>
o_5	<(c,5) (e,9) (h,20) (i,24)>
o_6	<(a,2) (d,7) (i,13)>

Table 6. Additional information for root node

Node	Count	IDs
c	4	o_2,o_3,o_4,o_5
i	4	o_1,o_3,o_4,o_5
e	3	o_2,o_4,o_5
d	2	o_1,o_3
g	2	o_1,o_3
a	1	o_1
f	1	o_2
h	1	o_5

TIP to extend $< a >$. Since node d has support 2 in Q, we extend $< a >$ to $< a \rightarrow d >$. By recursively calling the TIP algorithm, we obtain the path $< a \rightarrow d \rightarrow i >$ and the other paths are pruned.

Table 7. $< a >$-projected database $I_{<a>}$

ID	Observation
o_1	<(d,5) (g,10) (i,16)>
o_6	<(d,7) (i,13)>

Table 8. Frequent nodes in $I_{<a>}$

Node	Count
d	2
i	2
g	1

For node d, we update *PathSet* by calling the INSERT algorithm, as we observe from the prefix search tree in Figure 3 that node d has already been traversed in the previous mining result. Algorithm 4 gives the details of INSERT algorithm. The inputs are database D, node v, observation o, an integer k, *min_sup*, time threshold τ, and top-k maximal influential path set *PathSet*. The output is the set of top-k maximal influential paths *PathSet*. We first check whether observation o supports v (line 1). If o supports v, we update the *support* of v and $total\_score$ of path $< root \rightarrow \cdots \rightarrow v >$ and meanwhile update IDs of the supporting observations of node v (lines 2-5). For each child α of node v, if α is frequent, we recursively call the INSERT algorithm (lines 7-8). Otherwise, we call the TIP algorithm to explore branches that are pruned previously for possible top-k maximal influential paths (lines 9-15). Finally, we update the top-k maximal influential path set *PathSet* (lines 18-21) and *min_sup* (lines 22-23). With the insertion of observation o_6, we update the *support* of d to 3. Based on the additional information for node d as shown in Table 9, we know that the child node i is supported by o_6, so we update the *support* of i and meanwhile update the $total\_score$ of path $< d \rightarrow i >$.

For node i in observation o_6, as it is already traversed, we call the INSERT algorithm and update the *support* of i to 5. Figure 5 shows the prefix search tree constructed after inserting observation o_6. The bold lines represent the explored paths.

Algorithm 4. INSERT(D, v, o, k, min_sup, τ, $PathSet$)

Require: global variable $PathSet$, min_sup
Require: database D, node v, observation o, an integer k and time threshold τ
Ensure: Top-k maximal influential path set $PathSet$
1: **if** o supports v **then**
2: $support(v) = support(v) + 1$
3: let path $p = < root \rightarrow \cdots \rightarrow v >$
4: $total\_score(p) = total\_score(p) + score(p, o)$
5: add ID of o to IDs of the supporting observations of node v
6: **for each** child α of v **do**
7: **if** α is frequent **then**
8: **Call** INSERT(D, α, o, k, min_sup, τ, $PathSet$)
9: **else**
10: **if** o supports α **then**
11: let I be the set of observations in $D \cup \{o\}$ that support $< p \rightarrow \alpha >$
12: let $I_{<p \rightarrow \alpha>}$ be $< p \rightarrow \alpha >$-projected database
13: **Call** TIP($< p \rightarrow \alpha >$, $I_{<p \rightarrow \alpha>}$, k, min_sup, τ, $PathSet$)
14: **end if**
15: **end if**
16: **end for**
17: **end if**
18: let path $q \in PathSet$ such that $\nexists\, q' \in PathSet, total\_score(q') < total\_score(q)$
19: **if** $\exists\, p' \in T \setminus PathSet$ such that $total\_score(p') > total\_score(q)$ **then**
20: replace q with p'
21: **end if**
22: let path $q \in PathSet$ such that $\nexists\, q' \in PathSet, total\_score(q') < total\_score(q)$
23: $min\_sup = support(q)$
24: **return**

4.2 Delete Observation

Suppose we delete observation o_4: $< (c,3)(e,12)(i,36) >$ from the sample observation database D in Table 1. We update $PathSet$ by calling the DELETE algorithm. Algorithm 5 gives the details of DELETE algorithm. The inputs are database D, node v, observation o, an integer k, min_sup, time threshold τ, and top-k maximal influential path set $PathSet$. The output is the set of top-k maximal influential paths $PathSet$. We first scan the additional information table of node v to find every node α such that o supports α (line 1). For each node α, we update the $support$ of α and $total\_score$ of path $< root \rightarrow \cdots \rightarrow \alpha >$ and meanwhile update IDs of the supporting observations of node α (lines 3-6). We recursively call the DELETE algorithm on node α (line 7). After deleting observation o, we update the top-k maximal influential path set $PathSet$ (lines 9-12) and min_sup (lines 13-14). Finally, we call the TIP algorithm to explore branches that are pruned previously for possible top-k maximal influential paths (lines 15-20).

As observation o_4 is deleted from the sample database D, the support of node c, e and i will decrease. Note that we utilize the additional information for each node in the prefix tree as shown in Figure 4. Starting from the root node, based on the additional information for root node (Table 6), we know node c and i are supported by observation

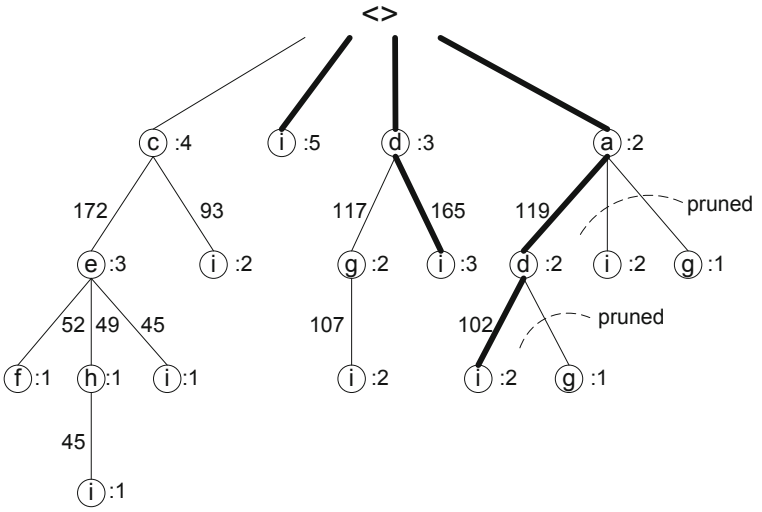


Fig. 5. Prefix search tree for new database after inserting observation o_6

o_4, so we decrease their support by 1. As for node c, based on its additional information (Table 10), the child e is also supported by o_4, so we update the *support* of node e and meanwhile update the $total\_score$ of path $< c \rightarrow e >$. Figure 6 shows the prefix search tree after deleting observation o_4. The bold lines represent the explored paths.

Table 9. Additional information for node d

Node	Count	IDs
g	2	o_1,o_3
i	2	o_1,o_3

Table 10. Additional information for node c

Node	Count	IDs
e	3	o_2,o_4,o_5
i	2	o_3,o_5
d	1	o_3
f	1	o_2
g	1	o_3
h	1	o_5

4.3 Complexity Analysis

In this section, we provide a brief analysis of the time and space complexity of TIP and IncTIP algorithms. The major cost of the TIP algorithm is the construction of projected databases. In the worst case, when no pruning takes place, TIP constructs a projected database for every observation in the database. Thus, both the worst-case time and space complexities are $O(N^L)$ where N is the number of tuples in the database and L is the maximum length of all observations. In addition, since we use pseudo-projection in our implementation, the space complexity can be reduced to the order of the size of the database.

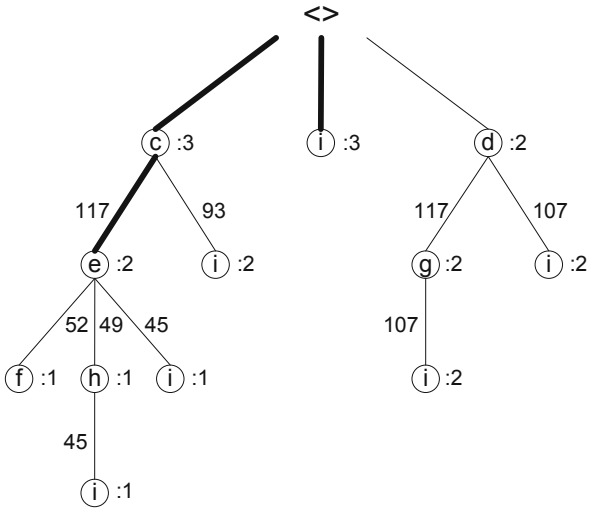


Fig. 6. Prefix search tree for new database after deleting observation o_4

Algorithm 5. DELETE(D, v, o, k, min_sup, τ, $PathSet$)

Require: global variable $PathSet$, min_sup
Require: database D, node v, observation o, an integer k and time threshold τ
Ensure: Top-k maximal influential path set $PathSet$
1: scan additional information table of node v, find every node α such that o supports α
2: **for each** node α **do**
3: $support(\alpha) = support(\alpha) - 1$
4: let path $p = < root \rightarrow \cdots \rightarrow \alpha >$
5: $total\_score(p) = total\_score(p) - score(p,o)$
6: remove ID of o from IDs of the supporting observations of node α
7: **Call** DELETE(D, α, o, k, min_sup, τ, $PathSet$)
8: **end for**
9: let path $q \in PathSet$ such that $\nexists\, q' \in PathSet, total\_score(q') < total\_score(q)$
10: **if** $\exists\, p' \in T \setminus PathSet$ such that $total\_score(p') > total\_score(q)$ **then**
11: replace q with p'
12: **end if**
13: let path $q \in PathSet$ such that $\nexists\, q' \in PathSet, total\_score(q') < total\_score(q)$
14: $min\_sup = support(q)$
15: scan additional information table of $root$, find every node v' that is not frequent
16: **for each** node v' **do**
17: let I be the set of observations in $D \setminus \{o\}$ that support v'
18: let $I_{v'}$ be v'-projected database
19: **Call** TIP(v', $I_{v'}$, k, min_sup, τ, $PathSet$)
20: **end for**
21: **return**

Similar to the TIP algorithm, the worst-case time complexity of IncTIP is $O(N^L)$ where N is the number of tuples in the database and L is the maximum length of all observations. For the IncTIP algorithm, we keep child node information for each node in the prefix tree to facilitate incremental mining. So the worst-case space complexity of IncTIP is $O((N+C)^L)$ where C is the number of child nodes for each node in the tree.

5 Experimental Evaluation

In this section, we conduct experiments to evaluate the effectiveness and efficiency of our proposed TIP and IncTIP algorithms. In the first set of experiments, we compare the TIP algorithm with the Naïve algorithm that finds the top-k influential paths without any optimization techniques. We also analyze the effectiveness of the two optimization strategies by implementing two versions of TIP, TIP^{early} and TIP^{pp}, where TIP^{early} utilizes only the early termination strategy without pseudo projection whereas TIP^{pp} utilizes only the pseudo projection technique without early termination. In the second set of experiments, we compare efficiency of TIP and IncTIP algorithms for incremental mining.

All algorithms are implemented in Java language. The experiments are performed using an Intel Core 2 Quad CPU 2.83 GHz system with 3GB of main memory and running Windows XP operating system.

We used one synthetic and two real-world datasets for performance evaluation. The OutbreakSim simulation model [22] is used to generate the synthetic dataset. This model mimics the real-world disease outbreak data in Western Australia. Our synthetic dataset consists of 48,507 outbreak cases for the South-west region of Western Australia over 100 days resulting in more than 150,000 observations.

Besides the synthetic dataset, we also utilize a real-world dataset, the MemeTracker data [13]. This MemeTracker dataset contains the quotes, phrases, and hyperlinks of the articles/blogposts that appear over prominent online news sites from August 2008 to April 2009. Each post contains a URL, time stamp, and all of the URLs of the posts it cites. Nodes are mostly news portals or news blogs and the time stamps in the data capture the time that a quote/phase was used in a post. Finally, there are directed hyperlinks among the posts. We use these hyperlinks to trace the flow of information. A site publishes a piece of information and uses hyperlinks to refer to the same or closely related pieces of information published by other sites. An observation is thus a collection of time-stamped hyperlinks among different sites that refer to the same or closely related pieces of information. We record one observation per piece – or closely related pieces – of information. We extract the most active media sites and blogs with the largest number of posts, and generate 46,352 observations.

Another real-world dataset is the Delicious dataset. The Delicious dataset contains social networking, bookmarking, and tagging information from a set of 20K users from Delicious social bookmarking system. Each user has bookmarks, tag assignments, i.e. tuples [user, tag, bookmark], and contact relations within the dataset social network. Each bookmark has a title and URL. The dataset also contains the timestamps when the tag assignments were done. We generate 19,230 observations from the Delicious dataset.

Table 11. Datasets Characteristics

Datasets	Cardinality	Avg Len	Max Len	Min Len
Synthetic	150,000	8.00	20	6
MemeTracker	46,352	13.72	42	3
Delicious	19,230	24.16	68	3

Table 11 shows the characteristics of the synthetic and real-world datasets used in the experiments including the number of input observations (Cardinality), average observation length (Avg Len), maximum observation length (Max Len) and minimum observation length (Min Len).

5.1 Efficiency Experiments

Efficiency of TIP. In this set of experiments, we evaluate the efficiency of TIP algorithm on both synthetic and real-world datasets. First, we vary the synthetic database size from 10k to 90k (See Figure 7). We set $k = 10$, time threshold $\tau = 100$, and radius of influence $\alpha = 1.0$. We observe that TIP algorithm remains efficient as the database size increases. In particular, the early termination optimization strategy is more effective in reducing the runtime compared to the pseudo projection.

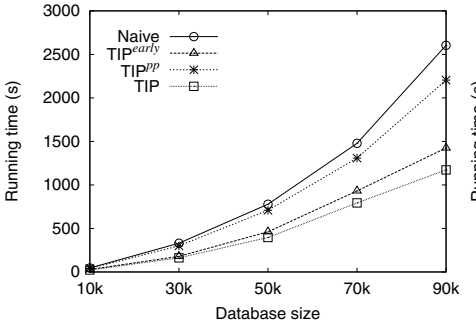


Fig. 7. Performance of varying database size on synthetic dataset

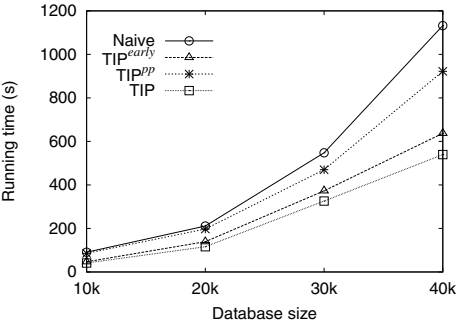


Fig. 8. Performance of varying database size on MemeTracker dataset

Similarly, for the real-world MemeTracker dataset, we generate the top-10 (i.e. $k = 10$) maximal influential paths by setting time threshold τ to 1000 and radius of influence α to 1.0. We randomly sample the dataset to vary the database size from 10k to 40k. As can be seen from Figure 8, TIP algorithm outperforms the Naïve algorithm with early termination playing a greater role in reducing the runtime of TIP.

Efficiency of IncTIP. We also evaluate the efficiency of IncTIP algorithm on both synthetic and real-world datasets. For the synthetic dataset, we set the size of original database D to 100k and vary the size of update database from 10k to 50k. We set $k = 5$, time threshold $\tau = 100$, and radius of influence $\alpha = 1.0$. Figure 9 shows the result. We observe that as the size of update database increases, the running time for both algorithms increases. However, IncTIP is more efficient than TIP. The reason is that each time when the database updates, TIP has to mine from scratch, but IncTIP only deals with the update part.

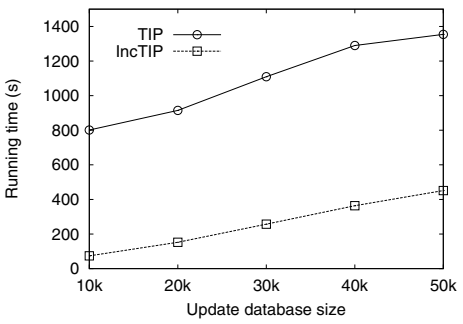


Fig. 9. Performance of varying update database size on synthetic dataset

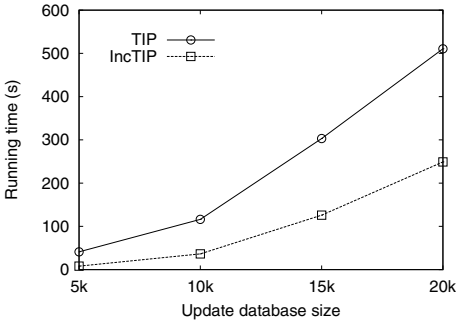


Fig. 10. Performance of varying update database size on MemeTracker dataset

For the MemeTracker dataset, we generate the top-10 (i.e. $k = 10$) maximal influential paths by setting time threshold τ to 1000 and radius of influence α to 1.0. We set the original database size to 25k and vary the size of update database from 5k to 20k. We can see from Figure 10 that IncTIP algorithm outperforms TIP algorithm as the size of update database increases.

We then compare the performance of IncTIP algorithm with an existing incremental mining algorithm IncSpan [29]. We evaluate IncTIP and IncSpan by varying update database size on both synthetic and real-world datasets. For both algorithms, we set the parameters such that they will generate the same number of patterns. Figure 11 shows the result on synthetic dataset by varying update database size from 10k to 50k. We can see that IncTIP outperforms IncSpan and the performance gap gets larger and larger as the update database size increases. This is because IncTIP utilizes time information to prune off the search space during mining process. Similar trend is observed for the MemeTracker dataset.

Memory Usage. Note that in order to facilitate incremental mining, we keep additional information for each node in the prefix tree. Thus, IncTIP algorithm will incur additional memory cost. In the experiments, we also compare the memory usage of TIP and IncTIP. Figure 13 shows the memory usage of TIP and IncTIP on the synthetic dataset. The size of original database D is 100k and the size of update database varies from 10k to 50k. We set $k = 5$, time threshold $\tau = 100$, and radius of influence $\alpha = 1.0$.

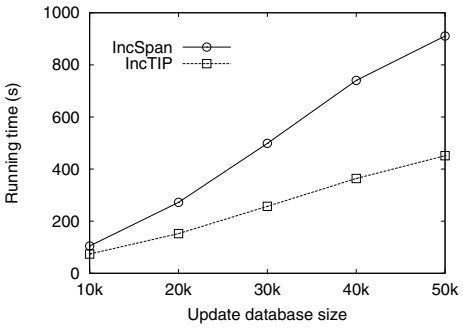


Fig. 11. Performance of varying update database size on synthetic dataset

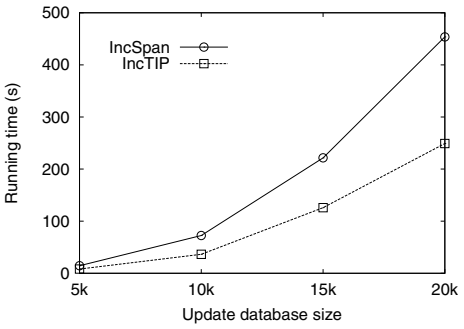


Fig. 12. Performance of varying update database size on MemeTracker dataset

We can see that as the update database size increases, the memory usage of both algorithms increases. However, IncTIP algorithm incurs more memory usage than TIP, as IncTIP keeps additional information to facilitate incremental mining.

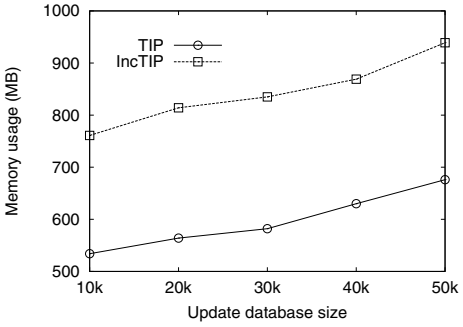


Fig. 13. Memory usage by varying update database size on synthetic dataset

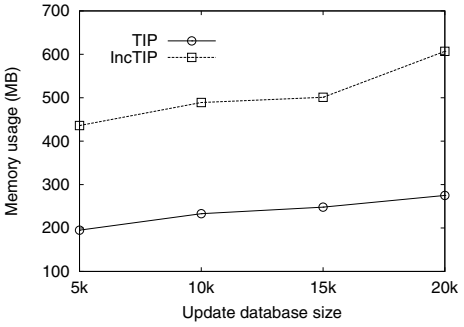


Fig. 14. Memory usage by varying update database size on MemeTracker dataset

The memory usage of TIP and IncTIP on the MemeTracker dataset is shown in Figure 14. We set the original database size to 25k and vary the size of update database from 5k to 20k. We set k to 10, τ to 1000 and α to 1.0. Similar trend can be observed for the MemeTracker dataset.

5.2 Sensitivity Experiments

Effect of k. Next, we investigate the effect of the number of maximal influential paths, k, on the performance of TIP algorithm. We set the database size to 20k and vary k from 5 to 25. Figure 15 shows the experimental results for the synthetic dataset. As can be seen, the runtime for both TIP and Naïve algorithm increases as k increases. However,

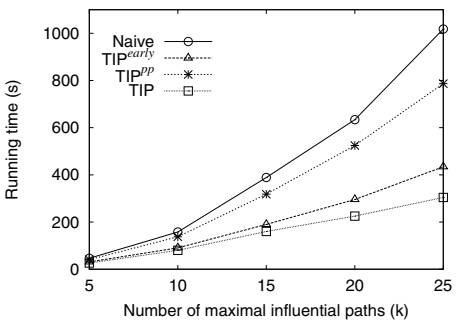


Fig. 15. Performance of TIP by varying k on 20k synthetic dataset

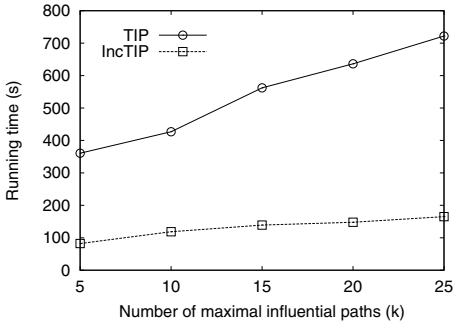


Fig. 16. Performance of IncTIP by varying k on 70k synthetic dataset

the runtime of TIP algorithm is half that of the Naïve algorithm demonstrating that TIP remains efficient even when k increases.

We also investigate the effect of the number of maximal influential paths, k, on the performance of IncTIP algorithm. For the synthetic dataset, we set the original database size to 50k, update database size to 20k and time threshold τ to 100. Figure 16 shows the runtime of IncTIP and TIP by varying k from 5 to 25. We can see that the runtime of both algorithms increases as k increases. However, IncTIP algorithm outperforms TIP algorithm by a large margin.

Effect of τ. Here, we examine the effect of varying the time threshold τ on the performance of TIP algorithm. Note that increasing τ is equivalent to increasing the search space, i.e. the number of potential influential paths. We set the number of maximal influential paths k to 5, database size $|D|$ to 20k and vary the time threshold τ from 10 to 50. Figure 17 shows that the runtime for all algorithms increases as τ increases. Similar trend is observed here with the TIP algorithm showing a significant reduction in runtime as compared to the Naïve algorithm. Similar trend is observed for the real-world datasets.

We also examine the effect of varying the time threshold τ on the performance of IncTIP algorithm. For the synthetic dataset, we set the original database size to 50k, update database size to 20k and k to 10. Figure 18 shows the runtime of IncTIP and TIP by varying τ from 10 to 50. We can see that the runtime of both algorithms increases as τ increases. However, IncTIP algorithm is more efficient than TIP algorithm for different values of τ. Similar trend is observed for the real-world datasets.

5.3 Effectiveness Experiments

Effectiveness of TIP. In the final set of experiments, we demonstrate the effectiveness of using maximal influential paths for prediction. To do cross validation, we partition the MemeTracker dataset into 4 folds (25% each). We use 75% of the total observations

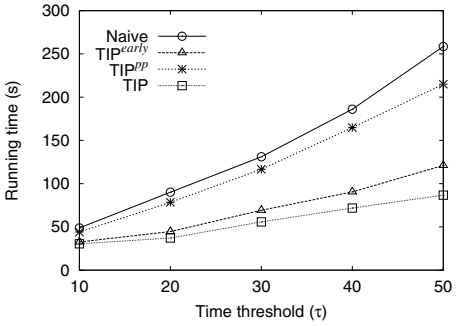


Fig. 17. Performance of TIP by varying τ on 20k synthetic dataset

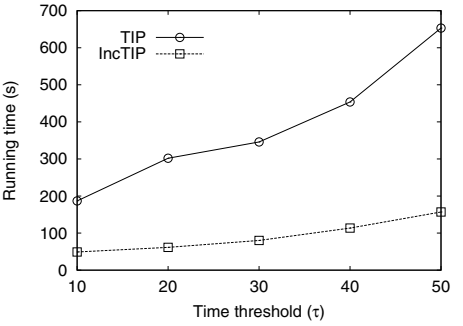


Fig. 18. Performance of IncTIP by varying τ on 70k synthetic dataset

for training and the remaining 25% for testing. We run the TIP algorithm on the training data to generate the top-k maximal influential paths. For each influential path $p = < v_1 \rightarrow v_2 \rightarrow \cdots \rightarrow v_{n-1} \rightarrow v_n >$ generated, we obtain the corresponding rule

$$r = \{< v_1 \rightarrow v_2 \rightarrow \cdots \rightarrow v_{n-1} > \Rightarrow < v_n >\}$$

with

$$confidence(r) = \frac{support(< v_1 \rightarrow v_2 \rightarrow \cdots \rightarrow v_{n-1} \rightarrow v_n >)}{support(< v_1 \rightarrow v_2 \rightarrow \cdots \rightarrow v_{n-1} >)}.$$

For each rule $< v_1 \rightarrow v_2 \rightarrow \cdots \rightarrow v_{n-1} > \Rightarrow < v_n >$, we determine the number of observations in the testing data that support $p' = < v_1 \rightarrow v_2 \rightarrow \cdots \rightarrow v_{n-1} >$. If there is at least one support observation in the testing data, we assign the probability of node v_n being influenced to the confidence of the rule, i.e. $\frac{support(p)}{support(p')}$. If we have more than one rule predicting that node v_n will be influenced, we assign the maximum confidence of the rules as the probability of node v_n being influenced.

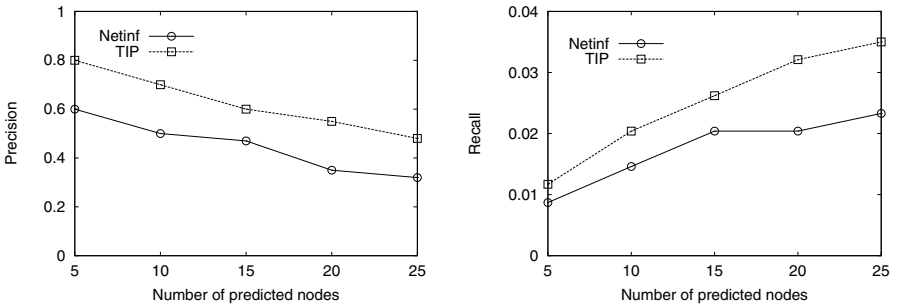
The set of predicted nodes are sorted in decreasing order of the probability of getting influenced. We consider a node to be the next influenced node if it is among the top-n nodes. Here top-n nodes are the first n non-duplicate nodes with highest probability of being influenced.

Let X be the set of nodes influenced in test data, and Y be the set of nodes predicted to be influenced in test data, then precision and recall are defined by the following equations:

$$precision = \frac{|X \cap Y|}{|Y|} \tag{6}$$

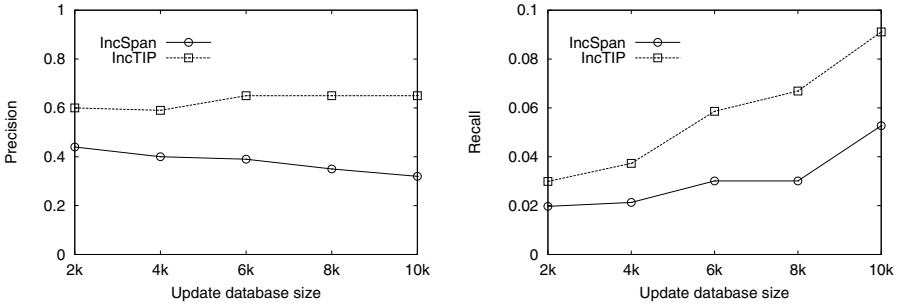
$$recall = \frac{|X \cap Y|}{|X|} \tag{7}$$

We compare the prediction accuracy of TIP algorithm with NetInf algorithm [7], which can only infer influential edge between two nodes. Similarly, we run NetInf algorithm



(a) Precision (b) Recall

Fig. 19. Precision and recall on MemeTracker dataset



(a) Precision (b) Recall

Fig. 20. Precision and recall on Delicious dataset

on the training data to generate a set of influential edges, say $< i \rightarrow j >$. We assign the probability of node j being influenced as $\frac{support(<i \rightarrow j>)}{support(<i>)}$.

We perform 4-fold cross validation for evaluating the prediction performance of both algorithms. Figure 19 shows the precision and recall results by varying the number of predicted nodes, n, from 5 to 25. We observe that TIP algorithm significantly outperforms NetInf algorithm for different values of n. This is because influential paths are more informative than influential edges and hence in predicting which node will be influenced next, the TIP algorithm tends to be more accurate than NetInf algorithm.

Effectiveness of IncTIP. We evaluate the effectiveness of IncTIP algorithm on the Delicious dataset. Note that due to space limitation, we do not report the efficiency results for the algorithms on the Delicious dataset. We partition the dataset into training data and testing data. The size of the training data varies from 2k to 10k. We set time threshold $\tau = 1000$, and radius of influence $\alpha = 1.0$. We run IncTIP on the training data to generate a set of rules and use the top-10 nodes for prediction. Similarly, we run IncSpan on the training data to generate a set of rules and select the top-10 predicted nodes.

We compare the prediction accuracy of IncTIP with IncSpan [29]. Figure 20 shows the precision and recall results by varying the size of update database (training data) from 2k to 10k. We observe that IncTIP outperforms IncSpan in both precision and recall measures. Further, the gap in both precision and recall between IncTIP and IncSpan widens as update database size increases. This demonstrates the effectiveness of IncTIP algorithm.

6 Related Work

In this section, we review works that are most relevant to our research. These include works in information diffusion analysis and incremental pattern mining.

6.1 Information Diffusion Analysis

Research on information diffusion analysis has focused on validating the existence of influence [6,3], studying the maximization of influence spread in the whole network [10,12,5,4], modeling direct influence in homogeneous networks [20], and mining topic-level influence on heterogeneous networks [16].

The works in [6,19] first study the influence maximization problem as an algorithmic problem. Kempe et al. [10] examine the influence maximization problem for a family of influence models. The authors design approximation algorithms for the independent cascade model. However, a drawback of their work is the efficiency issue of their greedy algorithm. Several recent studies try to address the efficiency issue by using new heuristics [12,11,5,4,18].

Gomez et al. [7] study the diffusion of information among blogs and online news sources. They assume that connections between nodes cannot be observed and use the observed cascades to infer a sparse, "hidden" network of information diffusion. They propose an iterative algorithm called NetInf which is based on submodular function optimization. NetInf first reconstructs the most likely structure of each cascade. Then it selects the most likely edge of the network in each iteration. The algorithm assumes that the weights of all edges have the same values.

Mathioudakis et al. [17] investigate the problem of sparsifying influence networks. Given a social graph and a list of actions propagating through it, they design the SPINE algorithm to find the "backbone" of the network through the use of the independent-cascade model [10]. SPINE has two phases: the first phase selects a set of arcs that yields a finite log-likelihood, while the second phase greedily seeks a solution of maximum log-likelihood. The effectiveness of SPINE came from its ability to reduce computation speed significantly.

In the field of sequence mining, Giannotti et al. [23] introduce a novel form of sequential pattern, called Temporally-Annotated Sequence (TAS), representing typical transition times between the events in a frequent sequence. They formalize the novel mining problem of discovering representative frequent TAS's as a combination of frequent sequential pattern mining and density-based clustering.

Information diffusion has also been considered from the view of the blogosphere, since it provides a unique resource for studying information flow. The works in [2,8] model and study the dynamics of diffusion of information in the blogosphere, while [9,3] design algorithms to identify influential blog posts and influential bloggers in a blogosphere.

Our work is different from the above methods. We do not require the underlying network structure to be known, and consider temporal information in information diffusion.

6.2 Incremental Pattern Mining

Works that are most relevant to our incremental mining method are in the field of incremental sequential pattern mining. Sequential pattern mining, first introduced in [24], is to find frequent subsequences from a sequence database. In many applications, databases are updated incrementally, which leads to the study of incremental mining of sequential patterns. Incremental sequential pattern mining methods can be classified into two categories, Apriori-based methods (e.g. ISM [26], ISE [28], and GSP+ [25]) and projection-based methods (e.g. IncSpan [29], IncSpan+ [30], PBIncSpan [31], and ISPBS [32]). Apriori-based incremental mining methods would generate huge set of candidate sequences, while projection-based incremental mining methods can avoid this by using pattern growth approach to mine sequential patterns.

Parthasarathy et al. [26] propose an incremental mining algorithm ISM based on SPADE [27], by maintaining a sequence lattice of the old database. The sequence lattice includes all the frequent sequences and a negative border. The negative border includes sequences that are infrequent but their subsequences are frequent. As the sequences in negative border do not necessarily have high support, it is very time and memory consuming to use negative border.

Masseglia et al. [28] develop another incremental mining algorithm ISE. ISE performs incremental pattern mining with a candidate generate-and-test approach – size-(k+1) candidates are generated from size-k frequent sequences. The problem of this algorithm is that it would generate a large number of candidates as well as multiple scans of the whole database. Zhang el al. [25] propose GSP+ algorithm for incremental sequential pattern mining when databases are updated by insertion or deletion. However, their method also belongs to candidate generate-and-test approach. Thus, it suffers from the same problems as ISE.

In [29], Cheng et al. propose an incremental mining algorithm, called IncSpan, by taking advantage of PrefixSpan [14]. IncSpan buffers a set of semi-frequent sequences for incremental mining. When a sequence database grows, the semi-frequent sequences have a higher probability to become frequent. Therefore, IncSpan can effectively reduce the number of database scan and projection. However, IncSpan cannot find the complete set of sequential patterns in the updated database [30].

Nguyen et al. [30] clarify the weakness of IncSpan by proving the incorrectness of the basic properties in IncSpan and propose a new algorithm called IncSpan+. IncSpan+ rectifies the shortcomings in generating the set of frequent sequential patterns and the set of semi-frequent sequential patterns.

In [31], Chen et al. argue that in general IncSpan+ cannot find complete set of sequential patterns, and propose a new incremental mining algorithm based on prefix tree, called PBIncSpan. PBIncSpan constructs a prefix tree to represent the sequential patterns and maintains the tree structure using width pruning and depth pruning when database updates. One problem of depth pruning is that it is based on Apriori property, so it is not very effective when the prefix tree is huge.

Our incremental mining method IncTIP is quite different from existing works on incremental pattern mining. We extend the pattern growth method with time constraint, and introduce a score function to measure different patterns.

7 Conclusion

In this paper, we develop a method for inferring top-k maximal influential paths which can truly capture the dynamics of information diffusion. Given a log of propagation observations of some information over a hidden network, our goal is to infer the top-k maximal influential paths that best explain these observations. We define a generative influence propagation model based on the Independent Cascade Model and Linear Threshold Model, which mathematically models the spread of certain information through a network. We design an algorithm called TIP to infer the top-k maximal influential paths. TIP utilizes the properties of top-k maximal influential paths to dynamically increase the support and prune the projected databases. In many applications, databases are updated incrementally. We also develop an incremental mining algorithm IncTIP to maintain the set of top-k maximal influential paths. We evaluate the proposed algorithms on both synthetic and real-world datasets. The experimental results demonstrate the effectiveness and efficiency of both TIP and IncTIP.

References

1. Rogers, E.: Diffusion of Innovations, 4th edn. Free Press (1995)
2. Adar, E., Adamic, L.A.: Tracking Information Epidemics in Blogspace. In: Web Intelligence, pp. 207–214 (2005)
3. Agarwal, N., Liu, H., Tang, L., Yu, P.S.: Identifying the Influential Bloggers in a Community. In: WSDM 2008, pp. 207–218 (2008)
4. Chen, W., Wang, C., Wang, Y.: Scalable Influence Maximization for Prevalent Viral Marketing in Large-Scale Social Networks. In: KDD 2010, pp. 1029–1038 (2010)
5. Chen, W., Wang, Y., Yang, S.: Efficient Influence Maximization in Social Networks. In: KDD 2009, pp. 199–208 (2009)
6. Domingos, P., Richardson, M.: Mining the Network Value of Customers. In: KDD 2001, pp. 57–66 (2001)
7. Gomez-Rodriguez, M., Leskovec, J., Krause, A.: Inferring Networks of Diffusion and Influence. In: KDD 2010, pp. 1019–1028 (2010)
8. Gruhl, D., Guha, R., Liben-nowell, D., Tomkins, A.: Information Diffusion through Blogspace. In: WWW 2004, pp. 491–501 (2004)
9. Java, A., Kolari, P., Finin, T., Oates, T.: Modeling the Spread of Influence on the Blogosphere. World Wide Web Conference Series (2006)
10. Kempe, D., Kleinberg, J., Tardos, É.: Maximizing the Spread of Influence through a Social Network. In: KDD 2003, pp. 137–146 (2003)

11. Kimura, M., Saito, K.: Tractable Models for Information Diffusion in Social Networks. In: Fürnkranz, J., Scheffer, T., Spiliopoulou, M. (eds.) PKDD 2006. LNCS (LNAI), vol. 4213, pp. 259–271. Springer, Heidelberg (2006)
12. Leskovec, J., Krause, A., Guestrin, C., Faloutsos, C., VanBriesen, J., Glance, N.: Cost-effective Outbreak Detection in Networks. In: KDD 2007, pp. 420–429 (2007)
13. Leskovec, J., Backstrom, L., Kleinberg, J.: Meme-tracking and the Dynamics of the News Cycle. In: KDD 2009, pp. 497–506 (2009)
14. Pei, J., Han, J., Mortazavi-Asl, B., Pinto, H., Chen, Q., Dayal, U., Hsu, M.-C.: Prefixspan: Mining Sequential Patterns Efficiently by Prefix-projected Pattern Growth. In: ICDE 2001, pp. 215–224 (2001)
15. Yan, X., Han, J., Afshar, R.: Clospan: Mining Closed Sequential Patterns in Large Datasets. In: SDM 2003, pp. 166–177 (2003)
16. Liu, L., Tang, J., Han, J., Jiang, M., Yang, S.: Mining Topic-level Influence in Heterogeneous Networks. In: CIKM 2010, pp. 199–208 (2010)
17. Mathioudakis, M., Bonchi, F., Castillo, C., Gionis, A., Ukkonen, A.: Sparsification of Influence Networks. In: KDD 2011, pp. 529–537 (2011)
18. Narayanam, R., Narahari, Y.: A Shapley Value-Based Approach to Discover Influential Nodes in Social Networks. IEEE T. Automation Science and Engineering 8(1), 130–147 (2011)
19. Richardson, M., Domingos, P.: Mining Knowledge-Sharing Sites for Viral Marketing. In: KDD 2002, pp. 61–70 (2002)
20. Tang, J., Sun, J., Wang, C., Yang, Z.: Social Influence Analysis in Large-scale Networks. In: KDD 2009, pp. 807–816 (2009)
21. Xu, E., Hsu, W., Lee, M.L., Patel, D.: Top-k Maximal Influential Paths in Network Data. In: Liddle, S.W., Schewe, K.-D., Tjoa, A.M., Zhou, X. (eds.) DEXA 2012, Part I. LNCS, vol. 7446, pp. 369–383. Springer, Heidelberg (2012)
22. Watkins, R., Eagleson, S., Beckett, S., Garner, G., Veenendaal, B., Wright, G., Plant, A.: Using GIS to Create Synthetic Disease Outbreaks. BMC Medical Informatics and Decision Making 7(1), 4 (2007)
23. Giannotti, F., Nanni, M., Pedreschi, D., Pinelli, F.: Mining Sequences with Temporal Annotations. In: SAC 2006, pp. 593–597 (2006)
24. Srikant, R., Agrawal, R.: Mining Sequential Patterns: Generalizations and Performance Improvements. In: Apers, P.M.G., Bouzeghoub, M., Gardarin, G. (eds.) EDBT 1996. LNCS, vol. 1057, pp. 3–17. Springer, Heidelberg (1996)
25. Zhang, M., Kao, B., Cheung, D., Yip, C.L.: Efficient Algorithms for Incremental Update of Frequent Sequences. In: Chen, M.-S., Yu, P.S., Liu, B. (eds.) PAKDD 2002. LNCS (LNAI), vol. 2336, pp. 186–197. Springer, Heidelberg (2002)
26. Parthasarathy, S., Zaki, M., Ogihara, M., Dwarkadas, S.: Incremental and Interactive Sequence Mining. In: CIKM 1999, pp. 251–258 (1999)
27. Zaki, M.: SPADE: An Efficient Algorithm for Mining Frequent Sequences. Machine Learning 42(1/2), 31–60 (2001)
28. Masseglia, F., Poncelet, P., Teisseire, M.: Incremental Mining of Sequential Patterns in Large Databases. Data & Knowledge Engineering 46(1), 97–121 (2003)
29. Cheng, H., Yan, X., Han, J.: IncSpan: Incremental Mining of Sequential Patterns in Large Database. In: KDD 2004, pp. 527–532 (2004)
30. Nguyen, S., Sun, X., Orlowska, M.: Improvements of IncSpan: Incremental Mining of Sequential Patterns in Large Database. In: Ho, T.-B., Cheung, D., Liu, H. (eds.) PAKDD 2005. LNCS (LNAI), vol. 3518, pp. 442–451. Springer, Heidelberg (2005)
31. Chen, Y., Guo, J., Wang, Y., Xiong, Y., Zhu, Y.: Incremental Mining of Sequential Patterns Using Prefix Tree. In: Zhou, Z.-H., Li, H., Yang, Q. (eds.) PAKDD 2007. LNCS (LNAI), vol. 4426, pp. 433–440. Springer, Heidelberg (2007)
32. Liu, J., Yan, S., Wang, Y., Ren, J.: Incremental mining algorithm of sequential patterns based on sequence tree. In: Lee, G. (ed.) Advances in Intelligent Systems. AISC, vol. 138, pp. 61–67. Springer, Heidelberg (2012)

Author Index