Inference of XML Integrity Constraints*

Matej Vitásek and Irena Mlýnková

Abstract. In this paper we expand upon the previous efforts to infer schema information from existing XML documents. We focus on inference of *integrity constraints*, more specifically `ID`/`IDREF`/`IDREFS` attributes in DTD. Building on the research by Barbosa and Mendelzon (2003) we introduce a heuristic approach to the problem of finding an optimal ID set. The approach is evaluated and tuned in a wide range of experiments.

1 Introduction

The XML is one of the leading formats for representing structured data. However, even though languages such as DTD and XML Schema [4] to describe XML structure exist for a long time, most of the documents use outdated or have no schema at all [17]. To tackle this problem one may employ reverse-engineering techniques to infer the schema from existing documents [1, 3]. In particular, [12] introduces the *jInfer* schema inference framework, dealing primarily with the structural parts of the schema: how all the elements, attributes and text data are to be organized in an XML document conforming to that schema.

But the structural requirements of a schema is not the only constraint that can be imposed on an XML document. The concept of *keys* and *foreign keys*, well known from the relational database world, applies to schemas as well and will be the topic of this paper.

From all the constraints that can be applied to an XML document by means of its schema, this paper will focus on keys and foreign keys. The most important concepts in this field are introduced in [5] and formalized in the notions of `ID`/`IDREF`/

Department of Software Engineering, Charles University in Prague, Czech Republic
e-mail: `vektor330@gmail.com`, `mlynkova@ksi.mff.cuni.cz`

* The paper was supported by the GAČR grant no. P202/10/0573.

T. Morzy et al. (Eds.): Advances in Databases and Information Systems, AISC 186, pp. 285–296.
springerlink.com

IDREFS attributes in DTD and XSD and xs:key/xs:keyref structures in XSD [4]. The scope of this paper is focussed on inference of ID/IDREF/IDREFS from existing XML documents. We discuss the equivalence of *ID set search* and *maximum weighted independent set*. We introduced the *mixed integer problem* and demonstrated how to solve it using external GLPK[1] solver in the environment of the *jInfer* framework [13].

The paper is structured as follows: In Section 2 we describe all necessary definitions and terms used in the rest of the text. In Section 3 we discuss the related papers and problems. In Section 4 we specify the MIP approach and in Section 5 we discuss related experiments. Finally in Section 6 we conclude.

2 Definitions

We use the representation introduced in [2], where an XML file is represented by a labeled tree consisting of nodes for elements, attributes and simple text data. Parent nodes are connected to child nodes with edges. This tree shall be called an *XML tree*. For a given node v of an XML tree we define $label(v)$ (name of the node in the document, only for elements and attributes), $id(v)$ (unique identifier across the document) and $value(v)$ (text content, only for attributes and simple text data). We denote $\mathscr{I}$ the set of all ids in the document. Without loss of generality we ignore the actual ordering of nodes in the tree.

From [4]: an XML attribute may have the type ID, IDREF or IDREFS. In short, ID attribute values must be unique across the document, element cannot have more than one ID attribute, IDREF and IDREFS values must reference an ID value.

Attribute Mapping

We will use a slightly different definition of *attribute mappings* (AMs) than [2] (without introducing keys from [5]) that will however give us the same result.

Definition 1 (Σ^E, Σ^A, Σ). *Σ^E is the* set of all element labels*, Σ^A is the* set of all attribute labels*. $\Sigma = \Sigma^E \cup \Sigma^A$ is their union and effectively the* set of all labels *in the document.*

Definition 2 (Attribute mapping). *For $x \in \Sigma^E$ and $y \in \Sigma^A$ we define the* attribute mapping *of y over x, denoted M_x^y, the $\mathscr{I} \times \mathscr{I}$ relation defined by*

$$M_x^y = \{(z,w) : label(z) = x, label(w) = y, parent(w) = z\}.$$

Thus the relation M_x^y contains edges in the XML tree connecting element nodes labeled x and attribute nodes labeled y. We can use a projection to retrieve all the unique *ids* of either elements or attributes from the relation, with notation $\pi_E(M_x^y)$ and $\pi_A(M_x^y)$.

[1] GNU Linear Programming Kit, http://www.gnu.org/s/glpk/

Definition 3 (Type of the attribute mapping). *AM M_x^y is of the* type $\tau(M_x^y) = x$.

Definition 4 (Image of attribute mapping). Image ι *of the attribute mapping M_x^y is defined as* $\iota(M_x^y) = \{z : z = value(w), w \in \pi_A(M_x^y)\}$.

So the image of an AM is a set of all the values of all attribute nodes contained in the mapping.

Definition 5 (*name*()). *Given an attribute mapping $m = M_x^y$, name(m) shall be defined as the string $x - y$.*

ID Set

Based on the requirements for an `ID` attribute [4] we define ID set with the help of the following definition.

Definition 6 (Candidate attribute mapping). *An attribute mapping m is a* candidate attribute mapping *if it is an injective function, that is,* $|m| = |\pi_E(m)| = |\pi_A(m)| = |\iota(m)|$.

Definition 7 (ID set). *A set of candidate attribute mappings $I = \{m_1, \ldots m_n\}$ is an* ID set *iff*

$$\bigcap_{m_i \in I} \tau(m_i) = \emptyset \text{ and } \bigcap_{m_i \in I} \iota(m_i) = \emptyset.$$

That is, an ID set has images without repeating values and all the types are unique (an element cannot have more than one `ID` attribute).

Given an ID set I, the requirements from [4] give us the following condition for an attribute mapping m to be marked `IDREF` (`IDREFS` in case of multivalued attributes): $\iota(m) \subseteq \bigcup_{m_i \in I} \iota(m_i)$.

Attribute Mapping Weight

This definition of weight for AMs or AM sets comes from [2] again. Let $M = \{m_1, \ldots m_i\}$ be the set of all non-empty AMs in the document.

Definition 8 (Support). Support *of AM m is defined as* $\phi(m) = \frac{|m|}{\sum_{p \in M} |p|}$.

Definition 9 (Coverage). Coverage *of AM m is defined as*

$$\chi(m) = \left(\sum_{p \in M, p \neq m} |\iota(m) \cap \iota(p)| \right) / \sum_{p \in M} |\iota(p)|.$$

Definition 10 (Weight). *For $\alpha, \beta \geq 0$ as relative priorities of support and coverage we define the AM* weight *as weight$(m) = \alpha.\phi(m) + \beta.\chi(m)$. For a set of AMs $S = \{m_1, \ldots m_i\}$ we define weight$(S) = \sum_{m \in S}$ weight(m).*

Definition 11 (Independent set). *Given an undirected graph $G = (V,E)$, a set of vertices $I \subseteq V$ is an* independent set*, iff* $\forall v_1, v_2 \in I, v_1 \neq v_2 : (v_1, v_2) \notin E$.

Definition 12 (Maximum weighted independent set). *Given an undirected graph $G = (V,E)$ and a weight function $w : V \to \mathbb{R}$, an independent set I_{max} is the* maximum weighted independent set*, iff the following is satisfied:*

$$\forall I' \subseteq V, I' \text{is an independent set} : \sum_{v \in I'} w(v) \leqslant \sum_{v \in I_{max}} w(v).$$

Finding the maximum weighted IS is an NP-hard optimization problem [15].

Linear Programming

The problem of *linear programming* is optimization of a linear function under a set of linear constraints. The formulation is usually called a *linear program* (LP). It can be written in the following form (a minimization version is possible, too).

$$\begin{aligned} \max_x z &= \mathbf{c}^{\mathrm{T}}\mathbf{x} \\ s.t. A\mathbf{x} &\leqslant \mathbf{b} \\ \mathbf{x} &\geqslant 0 \end{aligned}$$

where $\mathbf{x}$ is the vector of variables (to be found by the optimization), $\mathbf{b}$ is the vector and A its accompanying matrix of constraints and $\mathbf{c}$ is the vector of coefficients for the objective function.

Definition 13 (Mixed integer problem). *A* mixed integer problem *(MIP) is an instance of LP in which some or all variables are limited to integral or boolean values.*

While solving LP is usually possible in polynomial time using the *simplex algorithm* (see [6]), solving MIP in general is *NP*-hard.

3 Related Work

According to the article [2, Chapter 4], the problem of finding an ID set with weight more than some given K (K-IDSET) is in NP. Furthermore, the independent set (IS) problem can be reduced to K-IDSET, meaning K-IDSET is NP-hard and thus NP-complete. The article continues by proving that finding the maximum weighted IS can be reduced to the problem of finding an ID set with maximum weight (MAX-IDSET). This again means that MAX-IDSET is NP-complete and, furthermore, unless $P = NP$, MAX-IDSET has no constant factor approximation algorithm. The transformation works in both ways: it is equivalently possible to create a maximum weighted IS instance for a given MAX-IDSET instance. The authors then suggest a heuristic algorithm, which is incorporated into the framework proposed by this

paper. To the best of our knowledge, there are no other articles dealing with this problem.

Finding XML Keys

XML keys are a structure somewhat similar to `ID` attributes, but with a much larger expressive strength. They have been introduced in [5] and implemented in XML Schema[2].

Fajt in [8] summarizes several algorithms to help find XML keys in existing data, namely *Gordian*, *XML Primary Keys*, *SPIDER* and *DBA Companion*. Except for *XML Primary Keys*, they all are originally purposed to find keys in relational databases.

In our paper we find `ID` attributes directly. And even though we can always convert them to XML keys by the process mentioned above, we are unable to find more complex keys this way.

Maximum Weighted IS

Maximum weighted IS is a well researched topic with a lot of known direct or approximation algorithms, see e.g. [15] or [9]. According to [14], the best known approximation algorithm for weighted IS to-date achieves an approximation ratio of $3(\Delta+2)$, where Δ is the maximum degree of a vertex in the IS graph.

4 MIP Approach

In this section we introduce a new approach to finding maximum ID sets. First, we transform the problem formulation to maximum weighted IS problem formulation. Then we transform this into an MIP formulation and demonstrate how this can be solved using a solver such as GLPK. We will continue by applying heuristic approaches to improve the performance of the process.

4.1 ID Set to IS Formulation

Given $C = \{m_1, \ldots, m_n\}$ a set of all AMs in a document, we construct a graph $G = (V, E)$ as follows. For each AM $m_i \in C$ we create a vertex $v_{name(m_i)}$. Two vertices $v_{name(m_i)}$ and $v_{name(m_j)}$ shall be connected by an edge iff they cannot share the same ID set, either because they have the same type ($\tau(m_i) = \tau(m_j)$), or their images intersect ($\iota(m_i) \cap \iota(m_j) \neq \emptyset$). Weight of a vertex $v_{name(m_i)}$ is the weight of the attribute mapping: $w(v_{name(m_i)}) = weight(m_i)$.

[2] `http://www.w3.org/TR/xmlschema11-1/#Identity-constraint_Definition_details`

Now finding the maximum weighted IS in G finds the maximum (optimal) ID set in the original document.

4.2 IS to MIP Formulation

Given a graph $G = (V,E)$ with a weight function $w : V \to \mathbb{R}$, we introduce a binary variable x_i for each vertex $v_i \in V$ and an inequality constraint $x_i + x_j \leq 1$ for each edge $e = (v_i, v_j) \in E$. Furthermore we introduce an objective function in form $\sum_{x_i} x_i w(v_i)$.

It is obvious that the objective function and all the constraints consitute a MIP instance, and that solving it finds the maximum weigthed IS in G.

4.3 Finding ID Sets with GLPK

By chaining these two translations we can create a MIP formulation for a given set of AMs from a document. Solving this MIP instance will give us the optimal ID set for this document.

GLPK is a multi-platform, multi-purpose solver well suited for this task. This approach works and for any possible input we can let GLPK find the optimal solution. However, sometimes it takes too long to find the optimum, thus the aim of this paper is to improve this process.

A *heuristic* is an approach to problem solving based on prior experience, educated guess or common knowledge. A *heuristic algorithm* is one that, in a reasonably short time, generates a good, maybe even optimal solution to an optimization problem. However, it will not provide any formal guarantee about its quality.

While a heuristic algorithm can be seen as a tool designed to solve one specific problem, the notion of a *metaheuristic*[3] is a recipe to solve a whole family of problems.

Definition 14 (Solution Space, Solution Quality). Solution space *in general is the set of all permissible solutions (not violating any constraints). In the specific case of a MIP formulation it is the set of all* $\mathbf{x}$ *subject to* $A\mathbf{x} \leqslant \mathbf{b}$*. Every solution in the solution space has its* quality*, in case of MIP for solution* $\mathbf{x}$ *it is the value of the objective function in* $\mathbf{x}$*.*

Definition 15 (Solution Neighborhood). *A* neighborhood *of a solution* $\mathbf{x}$ *in the solution space are all the other solutions close to* $\mathbf{x}$ *according to some metric.*

The precise definition of the neighborhood is always adjusted according to specific needs. However, the neighborhood should be defined so that it is *continuous* (with respect to quality).

[3] A metaheuristic covers a wide range of topics in the field of heuristics, such as *Tabu Search* [10], *Ant Colony Optimization* [7] or *Genetic Algorithms* [11], to name a few.

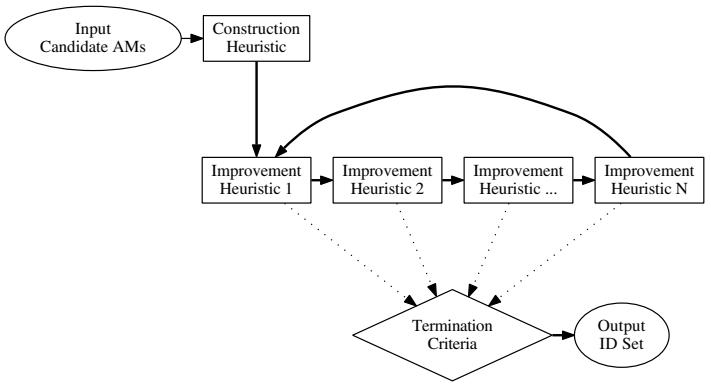


Fig. 1 Metaheuristic Schema

Definition 16 (Solution Pool, Incumbent Solution). Solution pool *(sometimes called* pool of feasible solutions *or* feasible pool*) is a set of solutions of different qualities that were found in one of the stages of the metaheuristic. The solution with the highest quality is called the* incumbent *solution.*

Our problem is the following: given a list of AMs with weights, find a non-conflicting subset maximizing the sum of weights in the subset. This sum will be the quality of the solution (subset). Our metaheuristic from Figure 1 works as follows: First we take the list of candidate AMs and ask a *construction heuristic* (CH) to provide us with a pool of solutions. Then, in a loop, we use this pool as input for *improvement heuristics* (IH) and in turns ask them to improve it. All the time we check whether *termination criteria* are met, and if so, we terminate the metaheuristic. The incumbent solution from the last pool is then the output ID set.

Constructions Heuristics

The purpose of a CH is to find at least one solution. Some IHs can profit from having several sub-optimal solutions, and some CHs can produce this pool from them. We implemented the following CHs:

- `FIDAX`: The deterministic algorithm described in [2] gives us one feasible solution.
- `Random`: Greedy algorithm, picks AMs at random and adds them if possible.
- `Fuzzy`: Greedy algorithm, similar to `Random`, but picks on weighted random.
- `Incremental`: Greedy algorithm, iterates AMs by descending weight, adds if possible.
- `Removal`: Greedy algorithm, adds all AMs, then iterates by ascending weight and removes until the ID set condition is not violated.

- `Glpk` (Truncated Branch & Bound): This is a time-constrained run of GLPK: limiting the run time gives us the best solution found so far. To be able to create a pool of solutions, the GLPK input is always shuffled. This causes the solver to explore the search tree in various orders, producing different solution in each run.

Improvement Heuristics

IHs start with a solution pool, attempt to improve one or more solutions in it and then return this improved pool. *Intensification* is the attempt to move the solution towards the nearby local optimum in the solution space. *Diversification* is the attempt to escape the local optimum, to be able to explore more of the solution space when the metaheuristic starts stagnating. A metaheuristic needs to combine both tendencies to explore the solution space and finally arrive at a local optimum.

We implemented the following IHs:

- `Remove Worst`: Removes the worst solution from the pool.
- `Random Remove`: Removes a random subset of specified size from each solution in the pool.
- `Hungry`: Intensification IH, tries to improve each solution in the pool by iteratively adding AMs sorted by weight, whenever possible.
- `Mutation`: Assumes that an incumbent solution already contains some AMs from the optimal solution. It takes a random guess and fixes some them, i.e. adds new constraints to the MIP formulation setting values of the respective variables to 1. This new formulation contains less free variables and is easier to solve, probably even to optimum. GLPK is run again with the constrained formulation. The solution found this way is a feasible solution of the original problem, but its optimum is not necessarily the same as in unconstrained formulation. It is intensification: we limit the search to the neighborhood of incumbent solution.
- `Crossover`: This IH looks for commonalities among the solutions in the pool. This is based on the idea that if more solutions agree on the same AMs, those probably are in the optimal solution, too. `Crossover` takes a parameter – fraction of solutions in the pool among which to look for commonalities. AMs found in every one of them are fixed in the modified MIP formulation the same way as in `Mutation`. This is again intensification.
- `Local Branching`: Another intensification IH, where the neighborhood being searched is defined by edit distance. The incumbent solution is represented by a vector of ones and zeroes, signifying which AMs belong to the ID set. A new constraint is added to the MIP formulation: for every other solution its edit distance, i.e. number of positions in which this solution and the incumbent solution differ, will have to be less than some threshold.

5 Experiments

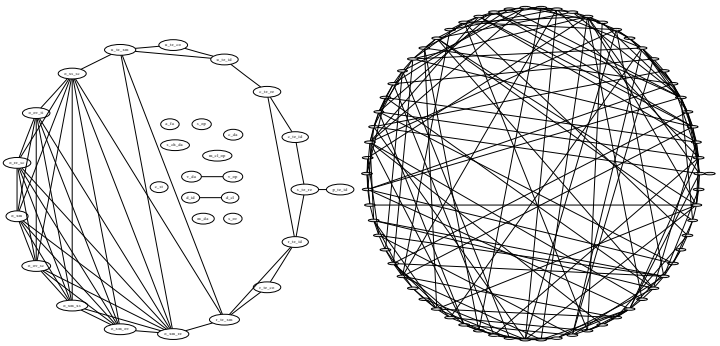
Our aim in performing experiments is: firstly, to describe behavior of the system and its components. Secondly, to evaluate performance in terms of speed and results quality. And finally, to find the "best" heuristic configuration.

In our experiments, we are using *realistic*, *converted* and *artificial* XML documents. Converted documents were created taking some of the values in text nodes and converting them to attributes. Artificial documents are randomly generated and have the following structure:

```
<graph>
  <vertex0 attr="-296887"/>
  <vertex1 attr="1729745"/>
  <vertex2 attr="-902054/>
  ...
  <vertex99 attr="-75459"/>

  <vertex82 attr="0"/><vertex21 attr="0"/>
  <vertex64 attr="1"/><vertex21 attr="1"/>
  <vertex44 attr="2"/><vertex2 attr="2"/>
  ...
  <vertex96 attr="99"/><vertex40 attr="99"/>
</graph>
```

It is interesting to look at graphs from the IS interpretation of our problem. Two of them are in Figure 2, one for a realistic XML document, one for an artificial. A quick reminder: vertices are AMs, edges connect pairs of AMs that cannot be in the same ID set together.



(a) *OVA3* (b) *70-245*

Fig. 2 IS Graphs for XML Documents

The size and density of these IS graphs determines how hard it is to solve the maximum ID set problem. It is obvious that the artificial document (*70-245*) is much harder – we created these artificial XML files to test our algorithms in the worst-case scenarios.

We performed a number experiments, but due to space limitations, we cannot possibly list them all here. Please refer to [16] for details. Only key findings will be mentioned here.

`Fuzzy` outperforms `FIDAX`

First, we compared `FIDAX` from the original article [2] to our trivial CHs `Random` and `Fuzzy`. Times for each of the algorithms were very similar. However, `FIDAX` was outperformed in terms of solution quality in most cases by our most trivial construction heuristics. Interestingly enough, adding our trivial IH `Hungry` after `FIDAX`, we were able to improve the quality of its results. This means, that `FIDAX` is not "greedy enough".

Best standalone CH is `Glpk`

We tried to find the CH that, on its own, produces the best results. The only CH with a parameter is `Glpk`, where we set the time limit to 1 second. It became obvious that pure `Glpk` is best one for this setup.

Best IH for `Glpk` is `Mutation`

After we found `Glpk` to be the best CH, we tried to find the best IH to complement it. This means, our metaheuristic setup for this case was $\texttt{Glpk}(limit = 1s) \rightarrow IH$, with the selected IH constrained for 1 run only. We found that the best IH for this purpose is `Mutation`, with time limit of 1 second (this was a common limit) and $fraction = 0.1$.

Chaining IHs – strategy

Next we tried the full strength of our metaheuristic approach by chaining several IHs together in "strategies" and comparing them. Our chosen CH was `Random` with pool size 10. The best strategy was the following.

$\texttt{RndRemove} \rightarrow \texttt{Mutation} \rightarrow \texttt{RndRemove} \rightarrow \texttt{Crossover} \rightarrow \texttt{RemoveWorst} \rightarrow \ldots < repeat >$.

In all IHs that support it, the fraction was 0.1 and the time limit 1 second. Results of several runs of this strategy are visualized in Figure 3. Each broken line is a representation of one run. X axis is time, Y axis is quality. Each break in the line indicates an IH finishing its work and handing the pool over to the next one, in a specific time and with a specific incumbent quality. Some lines terminate soon – this happens when the known optimum is reached.

Chaining IHs – tuning

We then tuned fractions in IHs in our "best" strategy. We found the best values: 0.5 for `RandomRemove`, 0.2 for `Mutation` and 0.1 for `Crossover`[4].

[4] Interpretation: `RandomRemove` fraction of 0.5 means that a randomly chosen half of all AMs from every ID set in the pool will be discarded. This amounts to a very strong diversification tendency and keeps the strategy from stalling in local optima. `Mutation` fraction of 0.2 means around $1/5^{th}$ of AMs in the incumbent solution will be fixed for the next GLPK optimization. `Crossover` fraction of 0.1 means that around 10% of ID sets in the pool (randomly chosen) will be scanned for common AMs.

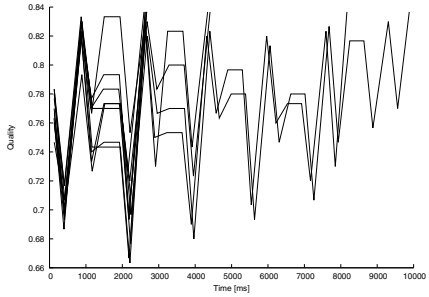


Fig. 3 Chained IHs – *100-100* Results

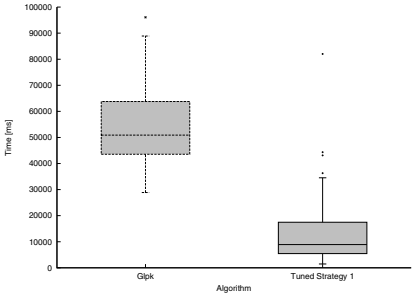


Fig. 4 Chained IHs – Pure `Glpk` vs. Tuned Strategy

Comparison with pure `Glpk`

Finally, we compared our tuned strategy to the performance of pure `Glpk`. We measured the time needed to find optimum in the hardest test XML document. The results are in Figure 4. Our tuned strategy found optimum in on average almost 4x shorter times than pure `Glpk` and under 10 seconds in more than a half of cases.

6 Conclusion

From all the integrity constraints in XML we chose the `ID`/`IDREF`/`IDREFS` attributes and decided to improve upon the search for them. We discussed the approach from [2] and the equivalence of ID set search and maximum weighted independent set. We introduced the MIP approach and demonstrated how to solve it using external GLPK solver in the environment of *jInfer* framework. Afterwards, we introduced a range of construction and improvement heuristics. We combined these algorithms to create a metaheuristic and performed a number of experiments to understand its behavior. Finally, we selected a promising metaheuristic strategy and tuned its parameters to find very good ID sets in short times.

A straightforward extension granting the ability to handle more than one input XML file has already been suggested in [2]. It is easy to add more CHs and IHs, as well as more metaheuristics using the existing IHs. A starting point is in [16]. It would be interesting to compare performance of our metaheuristic against two other interesting classes: Ant Colony Optimization and Genetic Programming.

References

1. Ahonen, H.: Generating Grammars for Structured Documents Using Grammatical Inference Methods. Ph.D. thesis, Department of Computer Science, University of Helsinki. Series of Publications A, Report A-1996-4 (1996)

2. Barbosa, D., Mendelzon, A.: Finding ID Attributes in XML Documents. In: Bellahsène, Z., Chaudhri, A.B., Rahm, E., Rys, M., Unland, R. (eds.) XSym 2003. LNCS, vol. 2824, pp. 180–194. Springer, Heidelberg (2003)
3. Bex, G.J., Neven, F., Vansummeren, S.: SchemaScope: a System for Inferring and Cleaning XML Schemas. In: Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data, SIGMOD 2008, pp. 1259–1262. ACM, New York (2008)
4. Bray, T., Paoli, J., Maler, E., Yergeau, F., Sperberg-McQueen, C.M.: Extensible Markup Language (XML) 1.0, 5th edn. W3C recommendation, W3C (2008)
5. Buneman, P., Davidson, S., Fan, W., Hara, C., Tan, W.C.: Keys for XML. In: Proceedings of the 10th International Conference on World Wide Web, WWW 2001, pp. 201–210. ACM, New York (2001)
6. Dantzig, G.: Linear Programming and Extensions. Landmarks in Physics and Mathematics. Princeton University Press (1998)
7. Dorigo, M., Stützle, T.: Ant Colony Optimization. Bradford Books. MIT Press (2004)
8. Fajt, S.: Mining XML Integrity Constraints. Master's thesis, Charles University in Prague (2010), http://www.ksi.mff.cuni.cz/projects/infer/keyminer/Fajt.pdf
9. Fomin, F.V., Grandoni, F., Kratsch, D.: A Measure & Conquer Approach for the Analysis of Exact Algorithms. J. ACM 56, 25:1–25:32 (2009)
10. Glover, F., Laguna, M.: Tabu Search. Kluwer Academic Publishers, Norwell (1997)
11. Goldberg, D.: Genetic Algorithms in Search, Optimization, and Machine Learning. In: Artificial Intelligence. Addison-Wesley Pub. Co. (1989)
12. Klempa, M., Mikula, M., Smetana, R., Švirec, M., Vitásek, M.: jInfer Architecture (2011), http://jinfer.sourceforge.net/modules/architecture.pdf
13. Klempa, M., Mikula, M., Smetana, R., Švirec, M., Vitásek, M.: jInfer: Java Framework for XML Schema Inference (2011), http://jinfer.sourceforge.net
14. Paschos, V.T.: A survey of Approximately Optimal Solutions to Some Covering and Packing Problems. ACM Comput. Surv. 29, 171–209 (1997)
15. Robson, J.: Algorithms for Maximum Independent Sets. Journal of Algorithms 7(3), 425–440 (1986)
16. Vitásek, M.: Inference of XML Integrity Constraints. Master's thesis, Charles University in Prague (2012), http://www.ksi.mff.cuni.cz/~mlynkova/dp/Vitasek.pdf
17. Vošta, O., Mlýnková, I., Pokorný, J.: Even an Ant Can Create an XSD. In: Haritsa, J.R., Kotagiri, R., Pudi, V. (eds.) DASFAA 2008. LNCS, vol. 4947, pp. 35–50. Springer, Heidelberg (2008)