





Polynomial Algorithms for Synthesizing Specific Classes of Optimal Block-Structured Processes

Costin Bădică¹  and Alexandru Popa^{2,3} 

¹ Department of Computers and Information Technology, University of Craiova, Craiova, Romania

cbadica@software.ucv.ro

² Department of Computer Science, University of Bucharest, Bucharest, Romania

alexandru.popa@fmi.unibuc.ro

³ National Institute for Research and Development in Informatics, Bucharest, Romania

Abstract. Synthesis of optimal business processes has practical applications in: manufacturing, scheduling, process mining, agent planning, and parallel computing. Block-structured models, in particular process trees, have certain advantages compared with other approaches regarding correctness and robustness. In this work we propose algorithms for the automated synthesis of optimal block-structured processes and then we perform a sound analysis of their correctness and complexity.

Keywords: Block structured processes · Combinatorial optimisation · Algorithms

1 Introduction

In process-centered applications (e.g. business and manufacturing, parallel computing, planning and scheduling) a natural goal is to perform the activities related to the business as quickly as possible. Based on domain-specific semantics, one can impose ordering constraints of the activities of a process. For example, if two activities are independent and there are enough resources to be allocated to each of them, then those activities can be scheduled for parallel execution. However, if an activity depends on the output produced by another activity, then the first activity can be scheduled for execution only after the completion of the second activity, i.e. their execution order is sequentially constrained.

There is a rich literature on process modeling in theoretical and applied computer science [13, 18]. Here we focus on block-structured models, in particular process trees, that are claimed to have certain advantages compared with other approaches [21].

A block-structured process is defined, informally, as a parallel or sequential composition of activities or other processes (we give a formal definition in Sect. 2). Each activity has an estimated duration of execution. The duration of two processes P and Q composed sequentially is $d(P) + d(Q)$, where $d(P)$ and $d(Q)$ is the duration of the process P and respectively, Q . Then, the duration of two processes P and Q composed in parallel is $\max\{d(P), d(Q)\}$, where $d(P)$ and $d(Q)$ is the duration of the process P and

respectively, Q . In this paper, we study the following problem: given a directed acyclic graph that specifies the ordering constraints on the activities, find a block structured process of minimum duration that satisfies the ordering constraints.

Motivation. The results of our work can be used in the area of business process management (BPM) with application in project scheduling [12]. BPM is a broad and well-established subject [7]. In this paper we focus on the specific problem of optimizing block structured processes that capture flexible project schedules satisfying a given set of activity precedence constraints [20].

In manufacturing there is interest for automated (i.e. using an algorithmic rather than manual) synthesis of correct process models, according to process-specific criteria like well-formedness, soundness and its variants (relaxed, weak and easy) [11]. Such correctness criteria can be ensured by synthesizing block-structured processes.

A possible approach uses standard project scheduling [12] and then synthesizes a block structured process from the schedule. However, this approach has drawbacks, as it can lead to unstructured and overly constrained processes. Moreover, as shown in [19], there are schedules that cannot be captured with a block structured process.

On the other hand, the block structured representation is a sort of template that satisfies problem constraints independently of activity durations, while a particular unstructured schedule does not always work if durations of activities unexpectedly change. Actually this can happen in realistic scenarios. For example, poorly performed work can increase duration of some activities. Moreover, activity durations are actually random variables, so in many applications, like those involving manual work, the exact value of the actual performance time depends on human performance and cannot be estimated exactly. This shows that block structured processes have obvious advantages over ordinary schedules. Therefore straightforward synthesis of block structured models is preferred and it can be based on ordering constraints [19,20] or on process mining [2,3,15].

Our work is also relevant for multiprocessor scheduling in parallel computing. For example, a new approach for parallel computing based on Series-Parallel Contention modeling was proposed by the early work of [25]. According to this approach, the parallel algorithm and the underlying machine are described as a series-parallel structured computation, similarly to a block-structured process.

Last but not least, block-structured processes are useful for developing sound intelligent distributed applications (e.g. scientific workflows) using multi-agent systems and concurrent plans, as proposed by Jason agent-oriented programming language [26].

Previous Work. Scheduling with ordering and resource constraints attracted computer science researches in the areas of multiprocessor and project scheduling. According to the early result of [24], scheduling with precedence constraints is NP-complete. Moreover, the problems of scheduling with precedence and resource constraints are included into the standard catalogue [8] of NP-complete problems (problems SS9 and SS10).

While planning and scheduling are classical problems, synthesis of block-structured processes from declarative specifications is a rather new problem that was only recently approached using heuristic algorithms [19,20]. These graphs, originally called ordering relation graphs, were first proposed and used to synthesize block-structured models in the works by Polyvyanyy et al. [22,23].

The SHAMASH knowledge-based approach for business process modeling and re-engineering was proposed in [1]. SHAMASH is claimed to be useful for process simulation and optimization (second goal is similar to ours). However, no evidence is provided, the work being focused on tool presentation, rather than its algorithmic foundation.

The problem of using automated planning in BPM, in particular for the automated design of template-based process models, was recently addressed by [16, 17]. Although this declarative approach guarantees correctness (understood as sound concurrency) and reusability (focusing on process templates), process optimization is not addressed by this approach. Nevertheless we claim that this is essential for business performance.

A considerable research effort was spent during the last decade for the synthesis of business process models from event logs [3]. A special attention was given to assuring the process correctness by focusing the Split Miner synthesis tool on producing block-structured or deadlock-free processes [2]. While not explicitly focused on the synthesis of optimal processes, experimental results revealed that the processes produced by Split Miner achieved considerably faster execution times than state-of-the-art methods.

A Greedy approach based on top down decomposition of the activity ordering graph was proposed in [4]. The approach was experimentally evaluated using two heuristics: hierarchical decomposition and critical path. An important result of [4] is that the hierarchical decomposition process (the basis for evaluating the hierarchical decomposition heuristic) satisfies the ordering constraints and it can be determined in polynomial time.

An exact solution based on declarative modeling using constraint logic programming was proposed in [5, 6]. However, the experimental evaluation revealed that this approach is feasible in practice only for small-size problems.

Our Results. We give a thorough study of block-structured process synthesis with activity ordering constraints.

First, since there is no known polynomial time exact algorithm for this problem, we study several variants in which the input graph is more restricted than an arbitrary directed acyclic graph. More precisely, we introduce a polynomial time algorithm that provides the optimum process (i.e. the one with the minimum duration) when the input graph is a tree (Sect. 3), a tree plus one edge (Sect. 4) and when the graph is bipartite (Sect. 5). The study of these particular cases is interesting from both practical and theoretical perspective. These particular classes were considered in the scheduling with precedence constraints setting by [14] (trees) and [10] (bipartite graphs) and, thus, these classes of graphs are relevant. In Sect. 6 we present a dynamic programming algorithm for solving the problem for general DAGs.

Beside the exact algorithms, we give polynomial time approximation algorithms for the problem. Due to space restrictions some of the proofs are omitted. However, the proofs, as well as some of our results are presented in the appendix (not included in the 13 pages version of the paper).

2 Preliminaries and Problem Definition

Let us consider a finite nonempty set Σ of activities. We focus on block structured process models that are defined as algebraic terms formed using sequential (\rightarrow) and parallel

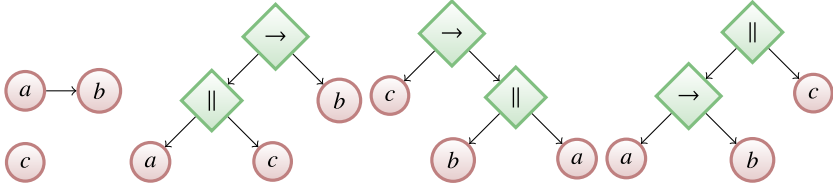


Fig. 1. From left to right: ordering graph \mathcal{G} , process P_1 , process P_2 , and process P_3

(\parallel) operators. The semantics of a process is defined as the set of admissible traces that contain exactly one instance of each activity. Sequential composition is interpreted as trace concatenation, while parallel composition is interpreted as trace interleaving.

Let $supp(P)$ be the support set of process P , representing the activities occurring in P . We denote activities of Σ with a, b, c, \dots and process terms with P, Q, R, \dots

Block structured processes are represented as tree structured terms (or process trees) defined as follows:

- If $a \in \Sigma$ then a is a process such that $supp(a) = \{a\}$.
- If P and Q are processes such that $supp(P) \cap supp(Q) = \emptyset$ then $P \rightarrow Q$ and $P \parallel Q$ are processes with $supp(P \rightarrow Q) = supp(P \parallel Q) = supp(P) \cup supp(Q)$.

The semantics of process P is given by its set of traces (language) $\mathcal{L}(P)$ as follows:

- $\mathcal{L}(a) = \{a\}$
- $\mathcal{L}(P \rightarrow Q) = \mathcal{L}(P) \rightarrow \mathcal{L}(Q)$
- $\mathcal{L}(P \parallel Q) = \mathcal{L}(P) \parallel \mathcal{L}(Q)$

Observe that if P is a well-formed block-structured process then all its traces $t \in \mathcal{L}(P)$ have the same length $|supp(P)|$.

We can impose ordering constraints of the activities of a process, based on domain-specific semantics. These constraints are specified using an activity ordering graph $\mathcal{G} = \langle \Sigma, E \rangle$ [20] such that:

- Σ is the set of nodes representing activities.
- $E \subseteq \Sigma \times \Sigma$ is the set of edges. Each edge represents an ordering constraint. If $(u, v) \in E$ then in each acceptable schedule activity u must precede v .

Observe that for an activity ordering graph $\mathcal{G} = \langle \Sigma, E \rangle$, set E defines a partial ordering relation on Σ , i.e. it is transitive and antisymmetric, so it cannot define cycles. In standard project scheduling terminology, graph \mathcal{G} is known as activity-on-node network [12] and it is a directed acyclic graph (DAG hereafter).

Let $\mathcal{G} = \langle \Sigma, E \rangle$ be an ordering graph and let t be a trace containing all the activities of Σ with no repetition. Then t satisfies \mathcal{G} , written as $t \models \mathcal{G}$, if and only if trace t does not contain activities ordered differently than \mathcal{G} specifies.

The language $\mathcal{L}(\mathcal{G})$ of an ordering graph \mathcal{G} is the set of all traces that satisfy \mathcal{G} , i.e. $\mathcal{L}(\mathcal{G}) = \{t \mid t \models \mathcal{G}\}$.

Let P be a process and let $\mathcal{G} = \langle \Sigma, E \rangle$ be an ordering graph. P satisfies \mathcal{G} written as $P \models \mathcal{G}$, if and only if:

- $\mathcal{L}(P) \subseteq \mathcal{L}(\mathcal{G})$, i.e. each trace of P satisfies \mathcal{G} , and
- $\text{supp}(P) = \Sigma$, i.e. all the activities of Σ occur in P .

Example 1. Figure 1 shows an ordering graph \mathcal{G} , and three processes P_1 , P_2 and P_3 . The total number of possible traces for the set of activities $\Sigma = \{a, b, c\}$ is $3! = 6$. Moreover, $\mathcal{L}(\mathcal{G}) = \{abc, acb, cab\}$. Observe also that $\mathcal{L}(P_1) = \{acb, cab\}$ and $\mathcal{L}(P_3) = \{cab, acb, abc\}$ showing that $P_1 \models \mathcal{G}$ and $P_3 \models \mathcal{G}$. However, as $\mathcal{L}(P_2) = \{cba, cab\}$, observe that $\mathcal{L}(P_2) \not\subseteq \mathcal{L}(\mathcal{G})$, so $P_2 \not\models \mathcal{G}$.

The set of processes P such that $P \models \mathcal{G}$ is nonempty, as it contains at least one sequential process defined by the topological sorting of \mathcal{G} .

Each activity has an estimated duration of execution represented using function $d : \Sigma \rightarrow \mathbb{R}^+$. The duration of execution $d(P)$ of a process P is defined as follows:

- If $P = a$ then $d(P) = d(a)$.
- $d(P \rightarrow Q) = d(P) + d(Q)$.
- $d(P \parallel Q) = \max \{d(P), d(Q)\}$.

The *minimum duration of execution* of a process that satisfies an ordering graph \mathcal{G} , denoted with $d_{OPT}(\mathcal{G})$, is: $d_{OPT}(\mathcal{G}) = \min_{P \models \mathcal{G}} \{d(P)\}$

An *optimal scheduling process* that satisfies a given ordering graph \mathcal{G} is a process OPT with a minimum duration of execution, i.e.: $OPT \models \mathcal{G}$, and $d(OPT) = d_{OPT}(\mathcal{G})$.

There is a finite and nonempty set of processes that satisfy an ordering graph \mathcal{G} , so an optimal process trivially exists. Moreover, as there is an exponential number of candidate processes satisfying \mathcal{G} , we postulate that the computation of the optimal process is generally an intractable problem.

Problem 1. For an activity ordering graph $\mathcal{G} = (\Sigma, E)$, find a process of minimum duration of execution satisfying \mathcal{G} .

In Subsect. 2.1 we define the *hierarchical decomposition process*, based on the hierarchical decomposition of the precedence DAG, initially introduced by [4]. We show that this process satisfies the problem constraints. The cost d_{HD} of this process can be computed in polynomial time. In Subsect. 2.2 we show a lower bound for the optimal solution of Problem 1.

2.1 Hierarchical Decomposition Process

Let $\mathcal{G} = \langle \Sigma, E \rangle$ be a precedence DAG.

- For each node $v \in \Sigma$ we define the set $I(v)$ of *input neighbors* of v as follows: $I(v) = \{u \in \Sigma \mid (u, v) \in E\}$.
- For each node $v \in \Sigma$ we define the *level* $l(v)$ of v as a function $l : v \rightarrow \mathbb{N}$ recursively constructed as follows:
 - If $I(v) = \emptyset$ then $l(v) = 1$.
 - If $I(v) \neq \emptyset$ then $l(v) = 1 + \max_{u \in I(v)} \{l(u)\}$.
- The *height* $l(\mathcal{G})$ of graph \mathcal{G} is defined as: $l(\mathcal{G}) = \max_{v \in V} \{l(v)\}$

- If $l = l(\mathcal{G}) \geq 1$ then the family of l sets $\{\Sigma_1, \Sigma_2, \dots, \Sigma_l\}$ defined as $\Sigma_i = \{v \mid l(v) = i\}$ for all $1 \leq i \leq l$ is a partition of Σ . If \mathcal{G}_i is the subgraph of \mathcal{G} induced by Σ_i then the family of graphs $\{\mathcal{G}_1, \mathcal{G}_2, \dots, \mathcal{G}_l\}$ is known as the *hierarchical decomposition* of \mathcal{G} .

Proposition 1. (*Hierarchical Decomposition Process*) Let $\mathcal{G} = \langle \Sigma, E \rangle$ be an ordering graph. The hierarchical decomposition process $HD(\mathcal{G})$ associated to \mathcal{G} is defined as:

- $P_i = \parallel_{v \in \Sigma_i} v$ for all $1 \leq i \leq l$.
- $HD(\mathcal{G}) = P_1 \rightarrow P_2 \rightarrow \dots \rightarrow P_l$.

Then $HD(\mathcal{G}) \models \mathcal{G}$.

Proof. Let t be a trace of process $HD(\mathcal{G})$. This means that for all activities u and v of t if u is before v in t then $l(u) < l(v)$. So there is a path from u to v in \mathcal{G} . It follows that $t \models \mathcal{G}$, i.e. $t \in \mathcal{L}(\mathcal{G})$. We conclude that $\mathcal{L}(HD(\mathcal{G})) \subseteq \mathcal{L}(\mathcal{G})$ that completes the proof. \square

Observe that the duration of execution $d_{HD}(\mathcal{G})$ of $HD(\mathcal{G})$ represents a non-trivial upper bound of the duration of execution of the optimal scheduling process $d_{OPT}(\mathcal{G})$, i.e. $d_{HD}(\mathcal{G}) \geq d_{OPT}(\mathcal{G})$.

2.2 Critical Path

Observe that an activity u cannot start unless all the neighboring activities from the input set $I(u)$ are finished. This time point is denoted with $start(u)$. Activity u that started at $start(u)$ will finish at time $finish(u) = start(u) + d(u)$. The values $start(u)$ and $finish(u)$ for each activity $u \in V$ can be computed using the *critical path method* [9]:

- If $I(u) = \emptyset$ then $start(u) = 0$ and $finish(u) = d(u)$.
- If $I(u) \neq \emptyset$ then $start(u) = \max_{v \in I(u)} \{finish(v)\}$ and $finish(u) = start(u) + d(u)$.

The maximum value of the finishing time of each activity, known as *critical path length*, is a lower bound for the duration of execution of the optimal scheduling process.

Proposition 2. (*Critical Path*) Let $\mathcal{G} = \langle \Sigma, E \rangle$ be an ordering graph and let $d_{CP}(\mathcal{G})$ be its critical path length. Then $d_{CP}(\mathcal{G})$ is a lower bound of the duration of execution of the optimal scheduling process $d_{OPT}(\mathcal{G})$, i.e. $d_{OPT}(\mathcal{G}) \geq d_{CP}(\mathcal{G})$.

Proof. This result follows from the definition of the critical path. From $OPT(\mathcal{G}) \models \mathcal{G}$ it follows that $OPT(\mathcal{G})$ executes sequentially the activities on the critical path so $d_{OPT}(\mathcal{G}) \geq d_{CP}(\mathcal{G})$. \square

3 An Exact Algorithm for Trees

In this section we show an exact algorithm for Problem 1, in the case when the activity graph is a *directed tree*. We define a directed tree to be a directed graph, where the root has indegree 0 and all the other nodes have indegree precisely 1. The algorithm presented in this section works also if we define a tree as a directed graph where the root has *outdegree* 0 and all other nodes have outdegree precisely 1.

The duration of the process returned by this algorithm is the longest path in the tree, thus matching the lower bound for the optimum. The algorithm is recursive and is presented in Algorithm 1.

```

ALG(Tree T)
Let r be the root of T and  $T_1, \dots, T_k$  be the subtrees of r.
if  $k = 0$  then
  | return r;
else
  | return  $r \rightarrow (ALG(T_1) \parallel \dots \parallel ALG(T_k))$ ;
end

```

Algorithm 1: Exact algorithm for a directed tree

Theorem 1. *ALG(T) returns the optimal solution for Problem 1 if T is a tree.*

Proof. The duration of the process returned by the algorithm is the duration of the longest path starting from the root. This is also a lower bound for the optimal solution as presented in the previous sections. Thus, the theorem follows. \square

4 An Exact Algorithm for a Tree Augmented with an Edge

In this section we present an exact algorithm for Problem 1 in case when the activity graph is a tree plus an extra edge.

Let r be the root of the tree. Let $y \in \Sigma$, be the *only* node that has indegree 2. Since the activity graph G consists of a tree plus one edge, then we know that y is unique. Then, let x_1 and x_2 be the two nodes such that (x_1, y) and $(x_2, y) \in E$. In order to process the subtree rooted at y we have to process the activities x_1 and x_2 .

We first explain a naive (and suboptimal) approach in order to gain intuition about our final algorithm. First, remove from G the subtrees rooted at x_1 and x_2 , run the exact Algorithm 1 for trees and then compose in parallel the subtrees rooted at x_1 and x_2 . The aforementioned approach may not produce the optimal process since, for example, the path from the root of G to x_1 may have longer duration than the path to x_2 . Thus, before processing x_2 , we may also process some part of the subtree of x_1 which is not a subtree of y . In turn, this subtree of x_1 may reduce the duration of the last step, when we process the subtrees of x_1 and x_2 in parallel.

We now present informally our algorithm (the formal definition is given by Algorithm 2). We first remove from the input graph the subgraph rooted at y . Then, let D_1 be the duration of path from the root to x_1 and D_2 the duration of the path from the root to x_2 . We select the largest subtree (with respect to the number of nodes) that generates a process with the duration less than $\max(D_1, D_2)$. This process is composed sequentially with the parallel composition of the remaining subtrees, including y (each of the remaining subtrees are processed according to Sect. 3).

In order to prove the correctness of Algorithm 3, we show the following lemma.

Lemma 1. *Consider an activity tree (i.e., an activity graph that is also a tree). Assume that we have a fixed budget B and the goal is to process as many activities as possible such that the duration of the process is at most B , and the constraints given by the activity graphs are preserved. The maximum set of nodes that can be processed given the budget B is unique and is obtained using Algorithm 3.*

Input: A graph G that consists of a tree T rooted at r plus one directed edge between two nodes.

1. Let y the *only* node with indegree 2. Let T_y be the subtree rooted at y . Create the tree T' from the input graph G , by removing the subtree T_y from G .
2. Let x_1 and x_2 be the two nodes such that (x_1, y) and $(x_2, y) \in E$.
3. Let D_1 be the length of the path from the root to x_1 and D_2 the length of the path from the root to x_2 .
4. Run Algorithm 3 on the tree T' and budget $\max(D_1, D_2)$. Let P_{budget} be the process returned by this algorithm.
5. Let T'_1, T'_2, \dots, T'_k , be the subtrees of T' that are *not yet processed* after the run of Algorithm 3. Let $P(A)$ be the process returned by the exact algorithm on trees from Section 3 on an input tree A .
6. The process returned by the algorithm is

$$P_{budget} \rightarrow (P(T'_1) \parallel P(T'_2) \parallel \dots \parallel P(T'_k) \parallel P(T_y))$$

Algorithm 2: Exact algorithm for a tree and an edge

```

Process(root  $r$ , budget  $B$ )
if  $d(r) > B$  then
    | return nil ;
else
    | Let  $r_1, \dots, r_k$ , be the roots of the subtrees of  $r$  for which  $\text{Process}(r_i, B - d(r)) \neq \text{nil}$  ;
    | return  $r \rightarrow (\text{Process}(r_1, B - d(r)) \parallel \dots \parallel \text{Process}(r_k, B - d(r)) )$  ;
end
    
```

Algorithm 3: Returns a process with maximum number of nodes in a tree given a fixed budget

Theorem 2. *Algorithm 2 returns the optimal process when the input graph is a tree plus one edge.*

Proof. In order to be able to process the activity y , and, therefore, the subtree of y , any process needs a duration of at least $\max(D_1, D_2)$. Thus, we aim to create a process with duration $\max(D_1, D_2)$ that uses as many nodes from the input graph G as possible (except, of course, the node y and nodes from its subtree). We create such a process using Algorithm 3 that takes as an input a budget and a tree and returns a process containing the maximum numbers of the activities from the tree that can be processed with the given budget. As we show in Lemma 1, there exists a unique set of nodes of maximum size that can be processed in a tree with a given budget. Moreover, this set of nodes, contains x_1 and x_2 and thus, we can process the rest of the subtrees. \square

5 An Exact Algorithm for Bipartite Graphs

We now introduce an exact algorithm for Problem 1 on bipartite graphs. We first present a recursive definition of the optimal process and we prove its correctness. Then we show

how this definition can be efficiently implemented by a top-down polynomial recursive algorithm using memoization.

Let U denote leftmost nodes and V denote rightmost nodes such that (U, V) is a partition of Σ . We sort nodes of U in non-decreasing order of their activity durations u_1, u_2, \dots, u_n . We denote with $U_i = \{u_1 \dots u_i\}$.

The idea is to consider two cases. If the undirected version of \mathcal{G} is not connected then the optimal process is a parallel composition of processes generated by the connected components (the proof, not shown here, is not difficult). Otherwise, if the undirected version of \mathcal{G} is connected then the optimal process is defined as the best process among a series of n processes $P_i, 1 \leq i \leq n$ defined as:

$$\begin{aligned}
 P_i &= (\parallel_{j=1}^i u_j) \rightarrow OPT(\Sigma \setminus U_i) \\
 d(P_i) &= d(u_i) + d_{OPT}(\Sigma \setminus U_i)
 \end{aligned}
 \tag{1}$$

With this observation, the recursive definition of an optimal process is:

$$OPT(\Sigma) = \begin{cases} a & \text{if } \Sigma = \{a\} \\ \max_{i=1}^k OPT(C_i) & \text{if } C_1, \dots, C_k, k \geq 2 \text{ are} \\ & \text{connected components of } \mathcal{G} \\ \arg \min_{P_i} d(P_i) & P_i \text{ defined by (1), otherwise} \end{cases}
 \tag{2}$$

The duration of the optimal process can be defined as:

$$d_{OPT}(\Sigma) = \begin{cases} d(a) & \text{if } \Sigma = \{a\} \\ \max_{i=1}^k d_{OPT}(C_i) & \text{if } C_1, \dots, C_k, k \geq 2 \text{ are} \\ & \text{connected components of } \mathcal{G} \\ \min_{i=1}^n d(P_i) & P_i \text{ defined by (1), otherwise} \end{cases}
 \tag{3}$$

Theorem 3. *The process defined by recursive Eqs. (2) and (3) is the optimal process on bipartite graphs.*

In order to obtain a polynomial algorithm based on Eqs. (2) and (3), first observe that this recursive process always generates a polynomial number of subsets of Σ . This result is stated by the following lemma.

Lemma 2. *Let us consider the recursive computational process determined by Eqs. (2) and (3). This process always generates $|U| + |V|$ subsets of Σ representing connected components.*

Proof. Observe that generated subsets representing connected components can be captured as a collection of trees with nodes labelled with subsets as follows. If \mathcal{G} is connected then there is a single tree with root Σ . Otherwise there is a collection of trees with roots labelled with connected components of \mathcal{G} . At each step we select for expansion one leaf of a tree labelled with set C that contains at least one element of U and we label it with $u_i \in U$ of minimum i . The children of C are connected components of graph with nodes $C \setminus \{u_i\}$. The process ends when no such set C can be selected. If a leaf node cannot be further expanded then it represents a singleton subset $\{v\} \subseteq V$ so we label it with v .

Finally we obtain a collection of trees such that each internal node is labelled with $u \in U$, each external node is labelled with $v \in V$ and each tree node has a unique label. So the number of nodes representing the subsets generated by the process is $|U| + |V|$. □

```

OPT(dag  $\mathcal{G} = \langle \Sigma = U \cup V, E \rangle$ , durations  $d$ )
if  $\Sigma = \{a\}$  then
  | return  $(a, d[a])$ ;
else if  $\mathcal{G}$  has connected components  $C_1, \dots, C_k$  s.t.  $k \geq 2$  then
  | for each  $i = 1, k$  do
  | |  $Opt_i \leftarrow Completed(C_i)$ ;
  | | if  $Opt_i = nil$  then
  | | |  $Opt_i \leftarrow OPT(C_i, d)$ ;
  | | |  $Completed(C_i) \leftarrow Opt_i$ ;
  | | end
  | end
  | return  $(\|_{i=1}^k Opt_i.P, \max_{i=1}^k Opt_i.dP)$ ;
else
  |  $dP \leftarrow +\infty$ ;
  | for each  $i = 1, |U|$  do
  | |  $(Q, dQ) \leftarrow OPT(\Sigma \setminus U_i, d)$ ;
  | |  $dQ \leftarrow dQ + d[u_i]$ ;
  | | if  $dQ < dP$  then
  | | |  $P \leftarrow \|_{u \in U_i} u \rightarrow Q$ ;
  | | |  $dP \leftarrow dQ$ ;
  | | end
  | end
  | return  $(P, dP)$ ;
end

```

Algorithm 4: Recursive algorithm with memoization to compute the optimal process for a bipartite graph

We are using approach i), resulting in Algorithm 4. The algorithm is using a collection *Completed* for saving the subsets $S \subseteq \Sigma$ representing connected components generated by Eqs. (2) and (3) for which the optimal process P and its cost dP have been computed as pairs (P, dP) . If $Completed(S) = nil$ then the computation for S has not been done yet. Otherwise $Completed(S) = (P, dP)$.

Theorem 4. *Algorithm 4 runs in $O(|U| \cdot (|U| + |V|)^2)$.*

Proof. When *OPT* is invoked the first time for each connected component, the call generates at most $|U|$ calls on the third **if** branch. Each such call goes through connected components determination in the second **if** branch, thus taking at most $|U| + |V|$ steps. As there are $|U| + |V|$ connected components it follows that the running time of Algorithm 4 is $O(|U| \cdot (|U| + |V|)^2)$. \square

6 An Exact Dynamic Programming Algorithm for Arbitrary DAG

Let $\mathcal{G} = \langle \Sigma, E \rangle$ be an ordering graph. If $S \subseteq \Sigma$ is a nonempty set then \mathcal{G}_S is the sub-graph of \mathcal{G} induced by S . The space of sub-problems is represented by all optimal sub-processes defined by nonempty subsets of Σ . A sub-process is optimal if it is either a singleton activity or it is composed of two sub-optimal processes.

We introduce arrays $Cost$ and $Proc$ indexed with nonempty subsets of Σ such that:

- $Cost[S]$ is the duration of the optimal sub-process with activities $S \subseteq \Sigma$
- $Proc[S]$ is the root of the optimal sub-process with activities $S \subseteq \Sigma$.

If S is a singleton then $Cost[S]$ and $Proc[S]$ are defined by Eq. 4.

$$\begin{aligned} Cost[\{a\}] &= d(a) \\ Proc[\{a\}] &= a \end{aligned} \tag{4}$$

If $|S| \geq 2$ then $Cost[S]$ and $Proc[S]$ are defined by Eq. 5.

$$\begin{aligned} C(S, \parallel) &= \min_{(X,Y) \models_{\parallel} \mathcal{G}_S} \max(Cost[X], Cost[Y]) \\ (X_{\parallel}, Y_{\parallel}) &= \arg \min_{(X,Y) \models_{\parallel} \mathcal{G}_S} \max(Cost[X], Cost[Y]) \\ C(S, \rightarrow) &= \min_{(X,Y) \models_{\rightarrow} \mathcal{G}_S} Cost[X] + Cost[Y] \\ (X_{\rightarrow}, Y_{\rightarrow}) &= \arg \min_{(X,Y) \models_{\rightarrow} \mathcal{G}_S} Cost[X] + Cost[Y] \\ Proc[S] &= (X_{\parallel}, Y_{\parallel}, \parallel) \text{ if } C(S, \parallel) \leq C(S, \rightarrow) \\ Cost[S] &= C(S, \parallel) \text{ if } C(S, \parallel) \leq C(S, \rightarrow) \\ Proc[S] &= (X_{\rightarrow}, Y_{\rightarrow}, \rightarrow) \text{ if } C(S, \rightarrow) < C(S, \parallel) \\ Cost[S] &= C(S, \rightarrow) \text{ if } C(S, \rightarrow) < C(S, \parallel) \end{aligned} \tag{5}$$

A dynamic programming algorithm takes the ordering graph \mathcal{G} and the array of activity durations d and computes matrices $Cost$ and $Proc$ following Eqs. 4 and 5. The optimal process can then be built using the information saved in array $Proc$.

Proposition 3. *If $(\Sigma_L, \Sigma_R) \models_{\parallel} \mathcal{G}$ then the undirected version of \mathcal{G} is not connected, so it can be partitioned into two or more connected components. So in this case if $\Sigma_1, \dots, \Sigma_k$ are its connected components with $k \geq 2$ then $P = P_1 \parallel \dots \parallel P_k$.*

According to the result of Proposition 3, the proposed dynamic programming algorithm works as follows. We first compute the connected components of the undirected version of \mathcal{G} . If there are $k \geq 2$ connected components then we will only consider the case when the optimal process is a parallel composition. Otherwise we will only consider the case when the optimal process is a sequential composition.

Let us now assume in what follows that the undirected version of \mathcal{G} is connected so in this case our optimal process is a sequential composition $L \rightarrow R$. We are interested to characterize sets Σ_L and $\Sigma_R = \Sigma \setminus \Sigma_L$ representing the alphabets of L and R .

Proposition 4. *Let us assume that the undirected version of \mathcal{G} is connected and let $P = L \rightarrow R$ a process. Let \mathcal{H} be the graph defined as the complement of the undirected version of \mathcal{G} , i.e. $\mathcal{H} = \overline{\mathcal{G}}$. For each $v \in \Sigma$ let us define $Lower(v) = \{u \mid (u, v) \in E \text{ or } u = v\}$. Then $P \models \mathcal{G}$ if and only if there exists a clique C of \mathcal{H} such that $\Sigma_L = \cup_{x \in C} Lower(x)$.*

Note that Proposition 4 can be used to make explicit the step of exploring the pairs of subsets (X, Y) such that $(X, Y) \models_{\rightarrow} \mathcal{G}_S$.

Now, using decomposition results stated by Propositions 3 and 4, our proposed dynamic programming approach is detailed by Algorithm 5.

```

RefOptProc(dag  $\mathcal{G}$ , durations  $d$ , matrix  $Cost$ , matrix  $Proc$ )
for  $a \in \Sigma$  do
  |  $Cost[\{a\}] \leftarrow d[a]$ ;
  |  $Proc[\{a\}] \leftarrow a$ ;
end
for  $i \leftarrow 2, |\Sigma|$  do
  | for each  $S \subseteq \Sigma$  s.t.  $|S| = i$  do
    | Let  $S_1, \dots, S_k$  be connected components of  $\overline{\mathcal{G}}_S$  if  $k \geq 2$  then
      | |  $Cost[S] \leftarrow \max_{i=1}^k Cost[S_i]$ ;
      | |  $Proc[S] \leftarrow (S_1, \dots, S_k, \parallel)$ ;
    | else
      | |  $C_{\rightarrow} \leftarrow +\infty$ ;
      | | for each clique  $C$  of  $\overline{\mathcal{G}}_S$  do
        | | |  $X \leftarrow \cup_{x \in C} Lower(x)$ ;
        | | |  $Y \leftarrow S \setminus X$ ;
        | | | if  $Cost[X] + Cost[Y] < C_{\rightarrow}$  then
          | | | |  $C_{\rightarrow} \leftarrow Cost[X] + Cost[Y]$ ;
          | | | |  $P_{\rightarrow} \leftarrow (X, Y, \rightarrow)$ ;
        | | | end
      | | end
      | |  $Cost[S] \leftarrow C_{\rightarrow}$ ;
      | |  $Proc[S] \leftarrow P_{\rightarrow}$ ;
    | end
  | end
end

```

Algorithm 5: Refined dynamic programming algorithm to compute the optimal process and its cost

7 Conclusions and Future Works

In this paper we provided several algorithms for the block-structured activity ordering problem. A natural open problem is to investigate the existence and design a polynomial time exact algorithm for the problem on arbitrary graphs or to show that the problem is NP-hard. We conjecture that the latter is true.

References

1. Aler, R., Borrajo, D., Camacho, D., Sierra-Alonso, A.: A knowledge-based approach for business process reengineering, shamash. *Knowl.-Based Syst.* **15**(8), 473–483 (2002)
2. Augusto, A., Conforti, R., Dumas, M., La Rosa, M., Bruno, G.: Automated discovery of structured process models from event logs: The discover-and-structure approach. *Data Knowl. Eng.* **117**, 373–392 (2018)
3. Augusto, A., et al.: Automated discovery of process models from event logs: review and benchmark. *IEEE Trans. Knowl. Data Eng.* **31**(4), 686–705 (2019)
4. Bădică, A., Bădică, C., Dănciulescu, D., Logofătu, D.: Greedy heuristics for automatic synthesis of efficient block-structured scheduling processes from declarative specifications. In: Iliadis, L., Maglogiannis, I., Plagianakos, V. (eds.) *AIAI 2018. IAICT*, vol. 519, pp. 183–195. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-92007-8_16

5. Bădică, A., Bădică, C., Ivanović, M., Logofătu, D.: Exploring the space of block structured scheduling processes using constraint logic programming. In: Kotenko, I., Badica, C., Desnitsky, V., El Baz, D., Ivanovic, M. (eds.) IDC 2019. SCI, vol. 868, pp. 149–159. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-32258-8_17
6. Bădică, A., Bădică, C., Ivanović, M.: Block structured scheduling using constraint logic programming. *AI Commun.* **33**(1), 41–57 (2020)
7. Dumas, M., La Rosa, M., Mendling, J., Reijers, H.A.: *Fundamentals of Business Process Management* (2nd ed.). Springer, New York (2018) <https://doi.org/10.1007/978-3-662-56509-4>
8. Garey, M.R., Johnson, D.S.: *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, New York (1979)
9. Kelley, J.E.: Critical-path planning and scheduling: mathematical basis. *Math. Oper. Res.* **9**(3), 296–320 (1961)
10. Kinne, J., Manuch, J., Rafiey, A., Rafiey, A.: Ordering with precedence constraints and budget minimization. *CoRR*, abs/1507.04885 (2015)
11. Klai, K., Desel, J.: Checking soundness of business processes compositionally using symbolic observation graphs. In: Giese, H., Rosu, G. (eds.) FMOODS/FORTE -2012. LNCS, vol. 7273, pp. 67–83. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-30793-5_5
12. Kolisch, R., Sprecher, A.: Psplib - a project scheduling library. *Eur. J. Oper. Res.* **96**(1), 205–216 (1997)
13. Kopp, O., Martin, D., Wutke, D., Leymann, F.: The difference between graph-based and block-structured business process modelling languages. *Enterp. Model. Inf. Syst. Architect.* **4**(1), 3–13 (2009)
14. Kumar, V.S.A., Marathe, M.V., Parthasarathy, S., Srinivasan, A.: Scheduling on unrelated machines under tree-like precedence constraints. *Algorithmica* **55**(1), 205–226 (2009)
15. Leemans, S.J.J., Fahland, D., van der Aalst, W.M.P.: Discovering block-structured process models from event logs - a constructive approach. In: Colom, J.-M., Desel, J. (eds.) PETRI NETS 2013. LNCS, vol. 7927, pp. 311–329. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38697-8_17
16. Marrella, A.: Automated planning for business process management. *J. Data Sem.* **8**(2), 79–98 (2019)
17. Marrella, A., Lespérance, Y.: A planning approach to the automated synthesis of template-based process models. *SOCA* **11**(4), 367–392 (2017)
18. Mili, H., Tremblay, G., Jaoude, G.B., Lefebvre, E., Elabed, L., El-Boussaidi, G.: Business process modeling languages: sorting through the alphabet soup. *ACM Comput. Surv.* **43**(1), 4:1–4:56 (2010)
19. Mrasek, R., Mülle, J., Böhm, K.: Automatic generation of optimized process models from declarative specifications. In: Zdravkovic, J., Kirikova, M., Johannesson, P. (eds.) CAiSE 2015. LNCS, vol. 9097, pp. 382–397. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-19069-3_24
20. Mrasek, R., Mülle, J., Böhm, K.: Process synthesis with sequential and parallel constraints. In: Debruyne, C., et al. (eds.) OTM 2016. LNCS, vol. 10033, pp. 43–60. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-48472-3_3
21. Pesic, M., van der Aalst, W.M.P.: A declarative approach for flexible business processes management. In: Eder, J., Dustdar, S. (eds.) BPM 2006. LNCS, vol. 4103, pp. 169–180. Springer, Heidelberg (2006). https://doi.org/10.1007/11837862_18
22. A. Polyvyanyy. Structuring process models. PhD thesis, University of Potsdam (2012)
23. Polyvyanyy, A., García-Bañuelos, L., Dumas, M.: Structuring acyclic process models. In: Hull, R., Mendling, J., Tai, S. (eds.) BPM 2010. LNCS, vol. 6336, pp. 276–293. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-15618-2_20

24. Ullman, J.D.: NP-complete scheduling problems. *J. Comput. Syst. Sci.* **10**(3), 384–393 (1975)
25. Gemund, A.J.C.: SPC: a model of parallel computation. In: Bougé, L., Fraigniaud, P., Mignotte, A., Robert, Y. (eds.) Euro-Par 1996. LNCS, vol. 1124, pp. 397–400. Springer, Heidelberg (1996). <https://doi.org/10.1007/BFb0024728>
26. Zatelli, M., Hübner, J.F., Bordini, R.H.: Concurrency in Jason. <http://jason.sourceforge.net/doc/tech/concurrency.html> (2016)