Greedy Heuristics for Automatic Synthesis of Efficient Block-Structured Scheduling Processes from Declarative Specifications

Amelia Bădică, Costin Bădică, Daniela Dănciulescu, Doina Logofătu

Greedy Heuristics for Automatic Synthesis of Efficient Block-Structured Scheduling Processes from Declarative Specifications

Amelia Bădică[1], Costin Bădică[1], Daniela Dănciulescu[1], and Doina Logofătu[2]

[1] University of Craiova, Romania,
ameliabd@yahoo.com,cbadica@software.ucv.ro,
[2] University of Applied Sciences, Frankfurt, Germany,
logofatu@fb2.fra-uas.de

Abstract. This paper introduces a new Greedy heuristic algorithm for the automatic synthesis of block-structured scheduling processes that satisfy a given set of declarative ordering constraints, as well as basic theoretical results that support the correctness of this algorithm. We propose two heuristics that can be used with this algorithm: hierarchical decomposition heuristic and critical path heuristic. We also present initial experimental results supporting the effectiveness and efficiency of our proposed algorithm and heuristics.

Keywords: Greedy algorithm, process model, ordering constraints, optimization

1 Introduction

There are many formalisms for the specification of business process models. Block-structured models have certain advantages compared with other approaches [3].

It is useful and quite intuitive to declaratively specify desired properties of process models. We are interested in constructing process models that are consistent with the given declarative specification [6]. This problem has practical applications in scheduling tasks encountered in manufacturing systems [4].

Manual construction of large process models satisfying a set of ordering constraints is almost impossible or at least not scalable. Automatic generation based on exhaustive exploration of the space of possibilities is difficult because of the huge number of potential candidates. The only feasible solution is to design automatic approaches based on efficient heuristic algorithms that are able to drastically prune the huge search space.

In this paper we focus on scheduling processes. In this case, the declarative specification defines the scheduling constraints. We are interested in determining optimal or at least, as efficient as possible, block-structured scheduling processes that satisfy the scheduling constraints. The optimization criterion requires the minimization of the total completion time. Optionally, we can add other constraints, like for example imposing upper bounds for the amount of parallel work. This constraint may result from the practical restriction regarding the limited availability of certain resources. In particular: i) our processes are defined only using sequential and parallel composition; ii) each activity must have exactly one instance in the schedule.

Our work was mainly influenced by previous results of [4, 5]. Nevertheless, our results are different in many aspects. Most important, our heuristics are deterministic and different. We are using the hierarchical decomposition of a graph, while [4, 5] are based on the more complex modular decomposition. We also provide theoretical results to support our work. Finally, we performed experiments with larger graphs, and our preliminary results suggest that our algorithm might be faster.

Note that there are many theoretical studies on evolutionary algorithms and randomized (meta/hyper) heuristics applied to combinatorial optimization algorithms [1]. Such works could be considered for the further expansion of our results by comparison of our method with different, but related approaches.

2 Process Models

Let us consider a finite nonempty set of activities Σ. A *trace* $t \in \Sigma^*$ is a sequence of zero or more activities [3]. The length of a trace $t = a_1 a_2 \dots a_n$ is n and this is denoted as $|t| = n$. The empty trace is denoted by ε and $|\varepsilon| = 0$. For each nonempty trace $t = a_1 a_2 \dots a_n$ we define: i) the head of t as $head(t) = a_1$, and ii) the tail of t as $tail(t) = a_2 \dots a_n$.

A *language* $L \subseteq 2^{\Sigma^*}$ is defined as a set of traces. We can define certain operations with languages. The *sequential composition* of two languages L_1 and L_2 denoted by $L_1 \to L_2$, is defined as follows:

$$L_1 \to L_2 = \{w = l_1 l_2 \mid l_1 \in L_1 \text{ and } l_2 \in L_2\}$$

This notation can be extended for a trace t and a language L as: $t \to L = \{t\} \to L$.

The *parallel composition* of two traces t_1 and t_2, denoted by $t_1 \parallel t_2$, is defined as:

- For each nonempty trace t we have: $t \parallel \varepsilon = \varepsilon \parallel t = \{t\}$
- For each nonempty traces t_1 and t_2 we have:

$$t_1 \to t_2 = (head(t_1) \to (tail(t_1) \parallel t_2)) \cup (head(t_2) \to (t_1 \parallel tail(t_2)))$$

The *parallel composition* $L_1 \parallel L_2$ of two languages L_1 and L_2 is now defined as:

$$L_1 \parallel L_2 = \cup_{t_1 \in L_1, t_2 \in L_2} t_1 \parallel t_2$$

Let us consider the set $\{\to, |, \|\}$ of three binary operators used for constructing block-structured processes. The operator $\to$ denotes *sequential composition*, the operator $|$ denotes the *nondeterministic choice*, and the operator $\|$ denotes *parallel composition*.

Let us denote with $a, b, c, \dots$ the activities of Σ and with $P, Q, R, \dots$ process terms. Process terms can be defined recursively as follows:

$$P ::= a \mid P \to Q \mid P \mid Q \mid P \parallel Q$$

The language $L(P)$ of process P is recursively defined as follows:

- $L(a) = \{a\}$

[3] Σ^* is the set of all sequences consisting of zero or more elements of Σ.

- $L(P \rightarrow Q) = L(P) \rightarrow L(Q)$
- $L(P \mid Q) = L(P) \cup L(Q)$
- $L(P \parallel Q) = L(P) \parallel L(Q)$

Operator $\parallel$ has higher precedence, operator $\rightarrow$ has middle precedence, and operator $\mid$ has lower precedence. All operators are associative, while $\parallel$ and $\mid$ are also commutative.

Process terms represent models of processes and they can be graphically depicted as trees or as block-structured flowcharts, as shown in Figure 1.

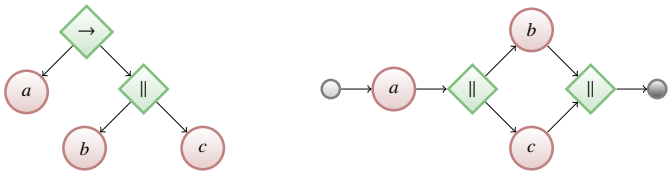

Fig. 1. Tree model of process $a \rightarrow b \parallel c$ (left) and its equivalent block-structured model (right)

In what follows we focus on process models with the following particularities:

- They represent sets of possible activity schedules. A schedule must contain exactly one instance of each activity.
- They use sequential ($\rightarrow$) and parallel ($\parallel$) operators. Scheduling processes are deterministic, explaining why nondeterministic choice is not used in their definition.

Rigorously defining scheduling processes requires the introduction of the support set $supp(P)$ of a process P that denotes the set of activities that occur in process P.

A *block-structured scheduling process* is recursively defined as follows:

- If a is an activity then a is also 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 $supp(P \rightarrow Q) = supp(P \parallel Q) = supp(P) \cup supp(Q)$.

For example, processes $a \parallel c \rightarrow b$ and $a \parallel (c \rightarrow b)$ are well-formed, and:

- $supp(a \parallel c \rightarrow b) = supp(a \parallel (c \rightarrow b)) = \{a, b, c\}$
- $L(a \parallel c \rightarrow b) = \{acb, cab\}$
- $L(a \parallel (c \rightarrow b)) = \{acb, cab, cba\}$

It is not difficult to observe that if P is a well-formed block-structured scheduling process then all its traces $t \in L(P)$ have the same length $|t| = |supp(P)|$.

3 Declarative Specification of Ordering Constraints

3.1 Activity Ordering Graph

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. there is a sequencing constraint between their execution order. Finally, if two activities define distinct action options then their execution is incompatible, so it cannot occur within the same schedule, i.e. they are mutually exclusive.

The ordering constraints imposed on each trace of a scheduling process are declaratively specified using an *activity ordering graph* $\mathcal{G} = \langle V, E \rangle$ [5] such that:

- V is the set of nodes and each node represents an activity.
- $E \subseteq V \times V$ is the set of edges. Each edge represents an ordering constraint. Set E is partitioned into two disjoint sets $E_{\rightarrow}$ and $E_{\neq}$ with the following meaning:
  - Set $E_{\rightarrow}$ specifies sequential ordering constraints. If $(u, v) \in E_{\rightarrow}$ then activity v cannot occur in a schedule without being preceded by activity u. $E_{\rightarrow}$ is a partial ordering, i.e. it is transitive and antisymmetric, so it cannot define cycles.
  - Set $E_{\neq}$ specifies mutual exclusion constraints. If $(u, v) \in E_{\neq}$ then activities u and v are incompatible, so they cannot occur within the same schedule. Set $E_{\neq}$ defines a symmetric relation.

Intuitively, satisfaction of mutual exclusion constraints requires the availability of nondeterministic choice operator in process definition. As we assumed that this operator is not available for scheduling processes, we will now focus only on sequential ordering constraints, i.e. we assume that $E_{\neq} = \emptyset$ so $E = E_{\rightarrow}$. This means that the ordering graph is a directed acyclic graph with arcs defining sequential ordering constraints.

If $t = a_1 a_2 \ldots a_n$ is a trace of a scheduling process and u, v are two activities of t then u precedes v in t, i.e. $u \overset{t}{\rightarrow} v$ if there are $1 \leq i < j \leq n$ such that $a_i = u$ and $a_j = v$.

Let $\mathcal{G} = \langle V, E \rangle$ be an ordering graph and let t be a trace containing all the activities of V with no repetition. Then t *satisfies* $\mathcal{G}$, written as $t \models \mathcal{G}$, if and only if $E_{\rightarrow} \subseteq \overset{t}{\rightarrow}$. This means that trace t cannot contain activities ordered differently than as specified by $\mathcal{G}$.

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

$$L(\mathcal{G}) = \{t \mid t \models \mathcal{G}\}$$

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

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

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 $\mathcal{G}$.

3.2 Optimal Scheduling Processes

Each activity has an estimated duration of execution that is represented using a 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 a given ordering graph $\mathcal{G}$, denoted with $d_{MIN}(\mathcal{G})$, is defined as:

$$d_{MIN}(\mathcal{G}) = \min_{P \models \mathcal{G}} \{d(P)\}$$

An *optimal scheduling process* that satisfies a given ordering graph $\mathcal{G}$ is a process P^* with a minimum duration of execution, i.e. it satisfies:

- $P^* \models \mathcal{G}$, and
- $d(P^*) = d_{MIN}(\mathcal{G})$.

There is a finite and nonempty set of processes that satisfy an ordering graph $\mathcal{G}$, so the optimal scheduling process trivially exists. Moreover, as there is an exponential number of candidate processes satisfying $\mathcal{G}$, we postulate that the computation of the optimal scheduling process is generally an intractable problem. Therefore, we will be focusing on developing efficient heuristic algorithms that are able to produce "suboptimal" or "good enough" scheduling processes using a reasonable computational effort.

4 Heuristics for Suboptimal Processes

We introduce two heuristics that are used to derive an efficient Greedy heuristic algorithm for computing a suboptimal scheduling process satisfying an ordering graph.

4.1 Hierarchical Decomposition Heuristic

Let $\mathcal{G} = \langle V, E \rangle$ be an ordering graph. Remember that $\mathcal{G}$ is a directed acyclic graph defining the sequential ordering constraints imposed on a scheduling process.

- For each node $v \in V$ we define the set $I(v)$ of *input neighbors* of v as follows: $I(v) = \{u \in V \mid (u, v) \in E\}$.
- For each node $v \in V$ we define the *level* $l(v)$ of v as a function $l : v \rightarrow \mathbb{N}$ such that:
  - If $I(v) = \emptyset$ then $l(v) = 0$.
  - 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 $m = l(\mathcal{G}) \geq 0$ then the family of $m + 1$ sets $\{V_0, V_1, \ldots, V_m\}$ defined as $V_i = \{v \mid l(v) = i\}$ for all $0 \leq i \leq m$ is a partition of V. If $\mathcal{G}_i$ is the subgraph of $\mathcal{G}$ induced by V_i then the family of graphs $\{\mathcal{G}_0, \mathcal{G}_1, \ldots, \mathcal{G}_m\}$ is known as the *hierarchical decomposition* of $\mathcal{G}$.

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

- $P_i = \|_{v \in V_i}\ v$ *for all* $0 \leq i \leq m$.
- $P_{HD}(\mathcal{G}) = P_0 \rightarrow P_1 \rightarrow \cdots \rightarrow P_m$.

Then $P_{HD}(\mathcal{G}) \models \mathcal{G}$. Moreover, the duration of execution of the hierarchical decomposition process associated to an ordering graph, denoted as $d_{HD}(\mathcal{G}) = d(P_{HD}(\mathcal{G}))$, represents a non-trivial upper bound of the duration of execution of the optimal scheduling process $d_{MIN}(\mathcal{G})$, i.e. $d_{HD}(\mathcal{G}) \geq d_{MIN}(\mathcal{G})$.

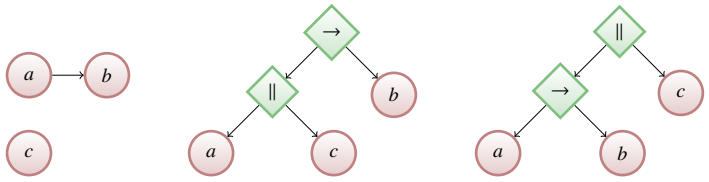

Fig. 2. Ordering graph $\mathcal{G}_1$ (left), process P_1 (middle) and process P_2 (right)

Figure 2 shows an ordering graph $\mathcal{G}_1$, and two processes P_1 and P_2 such that $\mathcal{G}_1 \models P_1$ and $\mathcal{G}_1 \models P_2$. The hierarchical decomposition of $\mathcal{G}_1$ is induced by the partition of its vertices $\{\{a, c\}, \{b\}\}$, so we can easily notice that P_1 is the hierarchical decomposition process of $\mathcal{G}_1$. Observe that:

- $d_{HD}(\mathcal{G}_1) = d(P_1) = \max\{d(a), d(c)\} + d(b)$
- $d(P_2) = \max\{d(a) + d(b), d(c)\}$

Clearly $d(P_1) \geq d(P_2)$ and P_2 is optimal (other satisfying processes are strictly sequential, incurring a higher duration of execution). But note that if $d(a) \geq d(c)$ then $d(P_1) = d(P_2) = d(a) + d(b)$ so the optimal scheduling process has duration $d_{HD}(\mathcal{G}_1)$ which shows that we can have equality in the inequality resulted from Proposition 1. However, if $d(a) < d(c)$ the optimal scheduling process has duration $d(P_2) = \max\{d(a) + d(b), d(c)\} < d_{HD}(\mathcal{G}_1) = d(c) + d(b)$.

4.2 Critical Path Heuristic

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* [2], as follows:

- 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 $\mathcal{G} = \langle V, E \rangle$ 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 $d_{MIN}(\mathcal{G})$, i.e. $d_{MIN}(\mathcal{G}) \geq d_{CP}(\mathcal{G})$.*

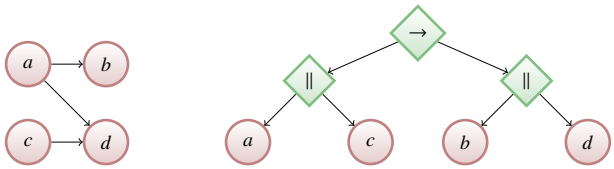

Fig. 3. Ordering graph $\mathcal{G}_2$ (left) and process P_3 (right)

Figure 3 shows an ordering graph $\mathcal{G}_2$ and its hierarchical decomposition process P_3. The critical path length of $\mathcal{G}_2$ is trivially $d_{CP}(\mathcal{G}_2) = \max\{d(a)+d(b), d(a)+d(d), d(c)+d(d)\}$, while $d(P_3) = \max\{d(a), d(c)\} + \max\{d(b), d(d)\} = \max\{d(a) + d(b), d(a) + d(d), d(c) + d(b), d(c) + d(d)\} = \max\{d_{CP}(\mathcal{G}_2), d(c) + d(b)\} \geq d(\mathcal{G}_2)$. Note that:

- If $d(c) + d(b) \leq \max\{d(a) + d(b), d(a) + d(d), d(c) + d(d)\}$ then $d_{CP}(\mathcal{G}_2) = d(P_3)$, i.e. the hierarchical decomposition process has a duration of execution equal to the critical path length. This clearly shows that $d_{MIN}(\mathcal{G}_2) = d_{CP}(\mathcal{G}_2)$.
- If $d(c) + d(b) > \max\{d(a) + d(b), d(a) + d(d), d(c) + d(d)\}$ then we infer that $d(c) > d(a)$, $d(b) > d(d)$, and $d(P_3) = d(b) + d(c)$. However, we do not know yet if in this case $d_{MIN}(\mathcal{G}_2)$ is equal to or strictly higher than $d_{CP}(\mathcal{G}_2)$. This depends on the other processes that satisfy $\mathcal{G}_2$. Two such processes are P_4 and P_5 (see Figure 4). Observe that if we choose $d(c)+d(d) > d(b)$ and $d(a)+d(b) > d(c)$ then $d_{CP}(\mathcal{G}_2) = \max\{d(a) + d(b), d(c) + d(d)\}$, while $d_{MIN}(\mathcal{G}_2) = \min\{d(P_3), d(P_4), d(P_5)\} = \min\{d(b) + d(c), d(a) + d(c) + d(d), d(a) + d(b) + d(d)\}$, that clearly shows that $d_{MIN}(\mathcal{G}_2) > d_{CP}(\mathcal{G}_2)$, i.e. the inequality from Proposition 2 is strict.

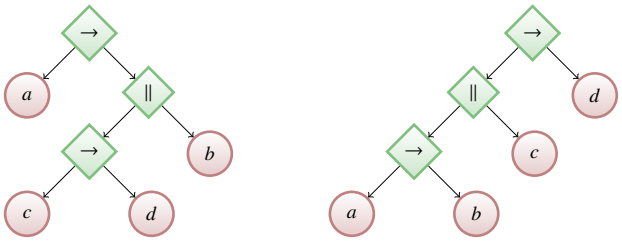

Fig. 4. Two other processes P_4 (left) and P_5 (right) that satisfy the ordering graph $\mathcal{G}_2$

4.3 Reducing the Duration of Execution

Analyzing Figure 2, we can observe that the duration of execution of the hierarchical decomposition process can be reduced by doing a transformation that pushes the parallel composition operations upper in the process tree. However, this transformation is

not always possible. We now provide sufficient conditions that enable the transformation and guarantee that the duration of execution of the resulted process is lower than of the original process. Referring at Figure 2, the key observation is that the set of nodes of graph $\mathcal{G}_1$ can be partitioned in two subsets $U_0 = \{a, b\}$ and $U_1 = \{c\}$ such that there are no arcs cross-linking nodes in U_0 to nodes in U_1 or nodes in U_1 to nodes in U_0. Note that such a decomposition is not possible for the graph $\mathcal{G}_2$ from Figure 2.

We consider the most general situation of reducing the duration of execution of a process $(P_1 \parallel P_2) \rightarrow (Q_1 \parallel Q_2)$. Similar results can be obtained for the processes of the form $(P_1 \parallel P_2) \rightarrow Q$ and $P \rightarrow (Q_1 \parallel Q_2)$.

Proposition 3. *(Reducing the Duration of Execution) Let $P = (P_1 \parallel P_2) \rightarrow (Q_1 \parallel Q_2)$ and let $\mathcal{G} = \langle V, E \rangle$ be an ordering graph such that $P \models \mathcal{G}$. Let us also assume that $((supp(P_1 \rightarrow Q_1) \boxtimes supp(P_2 \rightarrow Q_2)) \cap E = \emptyset$[4]. Then it follows that:*

- *Process $P' = (P_1 \rightarrow Q_1) \parallel (P_2 \rightarrow Q_2)$ is well-formed,*
- *$P' \models \mathcal{G}$, and*
- *$d(P) \geq d(P')$.*

4.4 Automatic Synthesis Algorithm

Let $\mathcal{G} = \langle V, E \rangle$ be an ordering graph and let $\mathcal{U}$ be the undirected graph obtained by removing the orientation of arcs of graph $\mathcal{G}$. We denote with $\mathcal{G}(W)$ and $\mathcal{U}(W)$ the subgraphs of $\mathcal{G}$ and $\mathcal{U}$ induced by a subset $W \subseteq V$ of nodes.

Let $\{V_0, V_1, \ldots, V_m\}$ be the partition of node set V defined by the hierarchical decomposition of $\mathcal{G}$. We define the following sets of nodes:

- $W_0 = V_0$
- $W_1 = W_0 \cup V_1$

  $\ldots$
- $W_m = W_{m-1} \cup V_m = V$

Let $\mathcal{U}_i = \mathcal{U}(W_i)$ and $\mathcal{G}_i = \mathcal{G}(W_i)$ for each $0 \leq i \leq m$. Each undirected graph $\mathcal{U}_i$ can be partitioned into connected components that induce the partition $\{U_1, U_2, \ldots, U_{k_i}\}$ of the set W_i of nodes such that $k_i > 1$. This situation is intuitively described in Figure 5.

Following the result of Proposition 3, the hierarchical decomposition process P defined for subgraph $\mathcal{G}_i$ can be transformed into process P' such that:

- $P' = \|_{j=1}^{k_i} P_j$
- $supp(P_j) = U_j$ for all $1 \leq j \leq k_i$
- $d(P) \geq d(P')$

Consider for example the sample ordering graph $\mathcal{G}_3$ from Figure 6. The partition of nodes corresponding to the hierarchical decomposition of $\mathcal{G}_3$ is $\{V_0, V_1, V_2\} = \{\{a, c\}, \{b, d\}, \{e\}\}$ and its height is $m = 2$. The hierarchical decomposition process of $\mathcal{G}_3$ is $P_6 = (a \parallel c) \rightarrow (b \parallel d) \rightarrow e$. Its duration of execution is 46.

We observe that for $i = 1$ the set $W_1 = V_0 \cup V_1 = \{a, b, c, d\}$ can be partitioned into $\{\{a, b\}, \{c, d\}\}$, so $k_1 = 2$. Using this observation we determine the transformed process

[4] Operator $\boxtimes$ denotes the symmetric cartesian product defined as $A \boxtimes B = (A \times B) \cup (B \times A)$

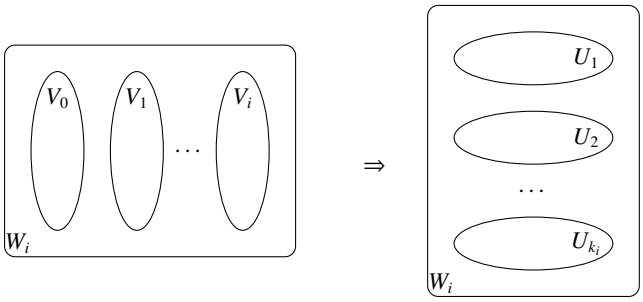

Fig. 5. Transformation to reduce duration of execution.

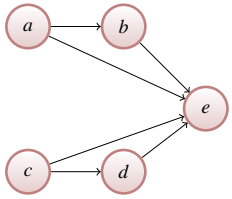

Fig. 6. Ordering graph $\mathcal{G}_3$. Activity durations: $d(a) = 10, d(b) = 18, d(c) = 20, d(d) = 7, d(e) = 8$.

$P_7 = ((a \to b) \parallel (c \to d)) \to e$. This process has the duration of execution of $36 < 46$. It follows that by applying our proposed transformation we were able to significantly reduce the duration of execution of process P_6 from $d(P_6) = 46$ to $d(P_7) = 36$.

We can combine this transformation with the hierarchical decomposition heuristic d_{HD} provided by Proposition 1 or with the critical path heuristic d_{CP} provided by Proposition 2 to design an efficient Greedy algorithm for the automatic synthesis of a suboptimal scheduling process that is consistent with a declarative specification.

Let $\mathcal{G} = \langle V, E \rangle$ be an ordering graph. The algorithm can be defined as a function $proc(W, \mathcal{G}(W))$ that takes a subset of nodes $W \subseteq V$, the subgraph $\mathcal{G}(W)$ of $\mathcal{G}$ induced by W and returns a suboptimal process that satisfies $\mathcal{G}(W)$.

Let $\{V_0, V_1, \ldots, V_m\}$ be the partition of node set V defined by the hierarchical decomposition of $\mathcal{G}$. Function $proc(V, \mathcal{G}(V))$ is recursively defined as follows:

- If $m = 0$ then $proc(V, \mathcal{G}(V)) = \|_{v \in V}\ v$.
- If $m > 0$ and $V_0 = \{v\}$ is a singleton set then $proc(V, \mathcal{G}(V)) = v \to proc(V \setminus \{v\}, \mathcal{G}(V \setminus \{v\}))$.
- If $m > 0$ and V_0 has at least two elements then for each $0 \leq i \leq m$ determine the number k_i of the sets of the partition of set W_i induced by the connected components of the undirected graph $\mathcal{U}_i$ obtained from the directed graph $\mathcal{G}_i$. We have $k_0 \geq k_1 \geq \cdots \geq k_m \geq 1$. Let i be the largest index for which $k_i > 1$. Such an index always exists as $k_0 = |V_0| > 1$. Select an index $0 \leq j \leq i$ for which the estimated duration of execution of the "synthesized process" (to be defined in what follows) is minimized.

We now recursively define the "synthesized process" and its estimated duration of execution, in terms of function *proc*. Let $\mathcal{G} = \langle V, E \rangle$ be an ordering graph, let $\{V_0, V_1, \ldots, V_m\}$ be the partition of node set V defined by the hierarchical decomposition of $\mathcal{G}$, and let us assume that $m > 0$ and $|V_0| > 1$. The "synthesized process" P_j and its estimated duration of execution $d_{G-EST}(P_j)$ with $EST \in \{HD, CP\}$ is:

- If $j = 0$ then $P_0 = \|_{v \in V_0} \rightarrow proc(V \setminus V_0, \mathcal{G}(V \setminus V_0))$ and is duration of execution is estimated to $d_{G-EST}(P_0) = \max_{v \in V_0}\{d(v)\} + d_{G-EST}(\mathcal{G}(V \setminus V_0))$.
- If $0 < j < m$ then let us consider the partition $\{Y_1, Y_2, \ldots, Y_{k_j}\}$ of W_j. Then $P_j = (\|_{i=1}^{k_j} proc(Y_i, \mathcal{G}(Y_i))) \rightarrow proc(V \setminus W_j, \mathcal{G}(V \setminus W_j))$ and is duration of execution is estimated to $d_{G-EST}(P_j) = \max_{i=1}^{k_j} \{d_{G-EST}(Y_i)\} + d_{G-EST}(V \setminus W_j)$.
- If $j = m$ then let us consider the partition $\{Y_1, Y_2, \ldots, Y_{k_m}\}$ of $W_m = V$. Then $P_m = \|_{i=1}^{k_m} proc(Y_i, \mathcal{G}(Y_i))$ and is duration of execution is estimated to $d_{G-EST}(P_m) = \max_{i=1}^{k_m} \{d_{G-EST}(Y_i)\}$.

Proposition 4. *(Duration of Execution of Greedy Suboptimal Processes) Let $d_{G-EST}(\mathcal{G})$ be the duration of execution of the suboptimal process that was computed with the Greedy algorithm using heuristic $EST \in \{HD, CP\}$. Then this process satisfies ordering graph $\mathcal{G}$ and $d_{HD}(\mathcal{G}) \geq d_{G-EST}(\mathcal{G}) \geq d_{MIN}(\mathcal{G}) \geq d_{CP}(\mathcal{G})$.*

5 Experimental Evaluation

We implemented our algorithm in Standard C using the 64-bit *GCC* compiler, version 5.1.0 and tested it on a x64-based PC with Intel(R) Core(TM) i7-5500U CPU at 2.40GHz running Windows 10. In this section we present the experimental results that we obtained with this implementation. The experiment was organized as follows:

- We randomly generated a number of directed acyclic graphs of increasing sizes representing ordering constraints, as well as random durations of execution for each activity of the graph. The parameters of a data set are: number n of graph nodes, number ng of generated graphs, minimum and maximum durations $dmin$ and $dmax$ of each activity, and the density factor $f \in [0, 1]$ of the graph. The higher is this factor the more dense is the graph. Value of f is given as a percentage.
- For each graph $\mathcal{G}$ we estimated the basic metrics given by the hierarchical decomposition heuristic $d_{HD}(\mathcal{G})$ and by the critical path heuristic $d_{CP}(\mathcal{G})$.
- For each graph $\mathcal{G}$ we computed the suboptimal scheduling process that satisfies $\mathcal{G}$ using the Greedy heuristic algorithm proposed in Section 4.4, in two variants: using the hierarchical decomposition heuristic and respectively using the critical path heuristic, to confirm the result claimed by Proposition 4, and to compare the results obtained for d_{G-HD} and d_{G-CP}.

The graph data sets were generated for the following values of the parameters: $ng = 100$, $n \in \{10, 50, 150, 300, 500, 700\}$, $dmin = 1$, $dmax = 20$, and density factor $f \in \{15\%, 30\%, 45\%, 60\%, 75\%\}$. For each test we recorded the total execution time and the values of the metrics of interest. We labelled each data set to reflect its number of nodes and density. For example if $n = 500$ and $f = 30\%$ then the label is 500-30.

# nodes / density	15%	30%	45%	60%	75%
10	0.019	0.009	0.012	0.010	0.012
50	0.191	0.202	0.186	0.208	0.198
150	1.813	1.898	2.057	2.157	2.295
300	7.883	9.358	10.552	10.482	12.435
500	26.200	31.889	36.889	41.378	46.221
700	65.121	75.967	90.575	104.268	116.500

Table 1. Total execution time in seconds for processing each data set

Table 1 presents the total execution time of running the synthesis algorithm for each data set. We observe that increasing the number of nodes, as well as the density, determines the increase of the execution time. Note that these times cover the processing of batches of 100 graphs. This means for example that the average time to process one graph of the 700-75 data set is approximately 1 second, i.e. our algorithm is quite fast.

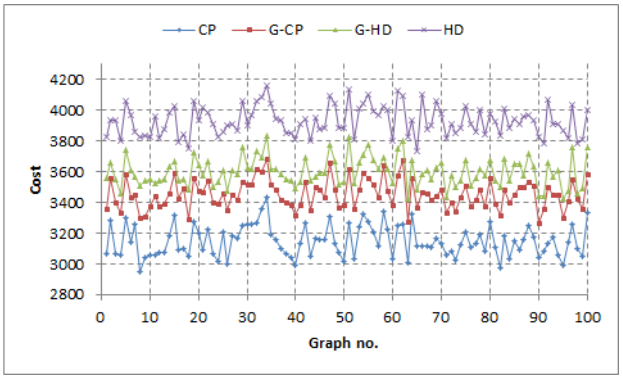

Fig. 7. Comparison of costs for the 700-30 data set

Figures 7 and 8 illustrate the values of the cost metrics for each graph of each data set 700-30 and 700-60. Three observations are drawn from these figures. Firstly, these experimental results are consistent with the theoretical results stated by Proposition 4. Secondly, that results of both experiments show that CP heuristic performs better than HD heuristic for almost all the graphs of the data set (there are few exceptions difficult to observe on the figures). Thirdly, the heuristics CP and HD tend to give closer results for higher density ordering graphs, as can be noticed by comparing the "closeness" of the cost values obtained for G-CP and G-HD for each data set 700-30 and 700-60.

6 Conclusions

We proposed a new Greedy algorithm for the automatic synthesis of block structured scheduling processes that satisfy given declarative ordering constraints. We presented

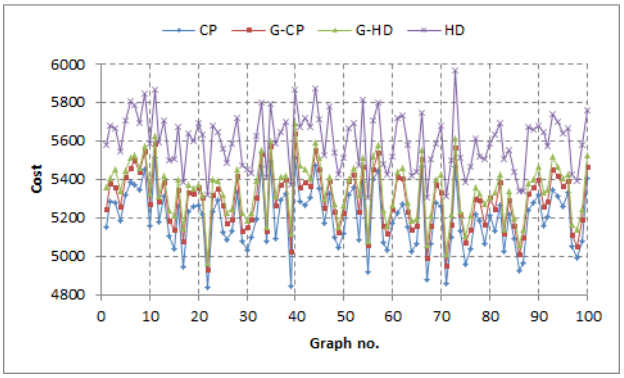

Fig. 8. Comparison of costs for the 700-60 data set

basic theoretical results that support the correctness of this algorithm. We proposed two heuristics that can be used with this algorithm: hierarchical decomposition and critical path. Our initial experimental results support the effectiveness of our proposals and suggest that the critical path heuristic performs better.

References

1. Demertzis, K., Iliadis, L.: Adaptive Elitist Differential Evolution Extreme Learning Machines on Big Data: Intelligent Recognition of Invasive Species. In: Angelov, P., Manolopoulos, Y., Iliadis, L., Roy, A., Vellasco, M. (eds.): Advances in Big Data. Advances in Intelligent Systems and Computing 529, 333–345. Springer (2017). doi:10.1007/978-3-319-47898-2_34
2. Kelley, J. E. Jr.: Critical-Path Planning and Scheduling: Mathematical Basis. Operations Research. 9, 3, 296–320. Informs (1961). doi:10.1287/opre.9.3.296
3. 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.): Application and Theory of Petri Nets and Concurrency. PETRI NETS 2013. LNCS 7927, 311–329. Springer (2013). doi:10.1007/978-3-642-38697-8_17
4. Mrasek, R., Mülle, J., Böhm, K.: Automatic Generation of Optimized Process Models from Declarative Specifications. In: Zdravkovic, J., Kirikova, M., Johannesson, P. (eds.): Advanced Information Systems Engineering. CAiSE 2015. LNCS 9097, 382–397. Springer (2015). doi: 10.1007/978-3-319-19069-3_24
5. Mrasek R., Mülle J., Böhm K.: Process Synthesis with Sequential and Parallel Constraints. In: Debruyne, C. et al. (eds.): On the Move to Meaningful Internet Systems: OTM 2016 Conferences. OTM 2016. LNCS 10033, 43–60. Springer (2016). 10.1007/978-3-319-48472-3_3
6. Pesic, M., van der Aalst, W.M.P.: A Declarative Approach for Flexible Business Processes Management. In: Eder, J., Dustdar, S. (eds.): Business Process Management Workshops. BPM 2006. LNCS 4103, 169–180, Springer (2006). doi:10.1007/11837862_18