Les Cahiers du GERAD ISSN: 0711–2440

An exact solution approach for the precedence-constrained class sequencing problem

R. Bürgy, P. Baptiste,
A. Hertz

G–2017–82

October 2017

La collection *Les Cahiers du GERAD* est constituée des travaux de recherche menés par nos membres. La plupart de ces documents de travail a été soumis à des revues avec comité de révision. Lorsqu'un document est accepté et publié, le pdf original est retiré si c'est nécessaire et un lien vers l'article publié est ajouté.

The series *Les Cahiers du GERAD* consists of working papers carried out by our members. Most of these pre-prints have been submitted to peer-reviewed journals. When accepted and published, if necessary, the original pdf is removed and a link to the published article is added.

Citation suggérée: Bürgy, Reinhard; Baptiste, Pierre; Hertz, Alain (Octobre 2017). An exact solution approach for the precedence-constrained class sequencing problem, Rapport technique, Les Cahiers du GERAD G-2017-82, GERAD, HEC Montréal, Canada.

Suggested citation: Bürgy, Reinhard; Baptiste, Pierre; Hertz, Alain (October 2017). An exact solution approach for the precedence-constrained class sequencing problem, Technical report, Les Cahiers du GERAD G-2017-82, GERAD, HEC Montréal, Canada

Avant de citer ce rapport technique, veuillez visiter notre site Web (https://www.gerad.ca/fr/papers/G-2017-82) afin de mettre à jour vos données de référence, s'il a été publié dans une revue scientifique.

Before citing this technical report, please visit our website (https://www.gerad.ca/en/papers/G-2017-82) to update your reference data, if it has been published in a scientific journal.

La publication de ces rapports de recherche est rendue possible grâce au soutien de HEC Montréal, Polytechnique Montréal, Université McGill, Université du Québec à Montréal, ainsi que du Fonds de recherche du Québec – Nature et technologies.

The publication of these research reports is made possible thanks to the support of HEC Montréal, Polytechnique Montréal, McGill University, Université du Québec à Montréal, as well as the Fonds de recherche du Québec – Nature et technologies.

Dépôt légal – Bibliothèque et Archives nationales du Québec, 2017
– Bibliothèque et Archives Canada, 2017

Legal deposit – Bibliothèque et Archives nationales du Québec, 2017
– Library and Archives Canada, 2017

GERAD HEC Montréal
3000, chemin de la Côte-Sainte-Catherine
Montréal (Québec) Canada H3T 2A7

Tél. : 514 340-6053
Téléc. : 514 340-5665
info@gerad.ca
www.gerad.ca

An exact solution approach for the precedence-constrained class sequencing problem

Reinhard Bürgy [a]

Pierre Baptiste [a]

Alain Hertz [a]

[a] *GERAD & Département de Mathématiques et de Génie Industriel, Polytechnique Montréal, Montréal (Québec), Canada H3C 3A7*

reinhard.burgy@gerad.ca
pierre.baptiste@polymtl.ca
alain.hertz@gerad.ca

October 2017
Les Cahiers du GERAD
G–2017–82

Copyright © 2017 GERAD, Bürgy, Baptiste, Hertz

Les textes publiés dans la série des rapports de recherche *Les Cahiers du GERAD* n'engagent que la responsabilité de leurs auteurs. Les auteurs conservent leur droit d'auteur et leurs droits moraux sur leurs publications et les utilisateurs s'engagent à reconnaître et respecter les exigences légales associées à ces droits. Ainsi, les utilisateurs:

- Peuvent télécharger et imprimer une copie de toute publication du portail public aux fins d'étude ou de recherche privée;
- Ne peuvent pas distribuer le matériel ou l'utiliser pour une activité à but lucratif ou pour un gain commercial;
- Peuvent distribuer gratuitement l'URL identifiant la publication.

Si vous pensez que ce document enfreint le droit d'auteur, contactez-nous en fournissant des détails. Nous supprimerons immédiatement l'accès au travail et enquêterons sur votre demande.

The authors are exclusively responsible for the content of their research papers published in the series *Les Cahiers du GERAD*. Copyright and moral rights for the publications are retained by the authors and the users must commit themselves to recognize and abide the legal requirements associated with these rights. Thus, users:

- May download and print one copy of any publication from the public portal for the purpose of private study or research;
- May not further distribute the material or use it for any profit-making activity or commercial gain;
- May freely distribute the URL identifying the publication.

If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

Abstract: This article discusses the precedence-constrainted class sequencing problem (PCCS). In scheduling terms, this problem models a one-machine problem with precedence constraints and setups where the goal is to minimize the number of setups. From a practical perspective, the PCCS problem can be used to address a wide range of applications such as, for example, decision problems in systems where multipurpose processors need retooling to switch from one operation to another.

Previous research has shown that the PCCS problem is difficult to solve from both a theoretical and computational perspective, and only little research has been conducted on computational methods. This article bridges this gap by proposing a branch-and-bound solution approach for the PCCS problem. Specialized reasoning, bounding, and heuristic procedures successfully exploit the problem's structure. Thus, the branch-and-bound approach solves large instances to optimality. The computational experiments further shed new light on what makes a PCCS instance difficult.

Keywords: Sequencing, setup times, precedence constraints, one-machine scheduling, branch-and-bound

Acknowledgments: R. Bürgy was partially supported by the Swiss National Science Foundation Grant P2FRP2_161720, which is gratefully acknowledged.

1 Introduction

Given are some operations, each belonging to some class, and precedence constraints among operations. The operations must be performed sequentially in a one-machine environment, and a setup is required between the execution of two consecutive operations if they belong to different classes. The precedence-constrained class sequencing (PCCS) problem asks for sequencing the operations so as to minimize the number of setups while respecting precedence constraints.

More formally, given are a set V of n operations, a set $\mathcal{C}$ of classes, and a set $A \subseteq V \times V$ of precedence constraints. Each operation $v \in V$ belongs to exactly one class $c_v \in \mathcal{C}$. Let $C = \{c_v : v \in V\}$ be the class assignment vector. An instance of the PCCS problem will be denoted by $(V, A, \mathcal{C}, C)$. Consider the directed graph $G = (V, A)$, where each operation $v \in V$ is represented by a node v and each precedence constraint $(v, w) \in A$ is modeled by an arc (v, w). Graph G will be called *precedence graph*. A solution to the PCCS problem is a total ordering of the operations that satisfies all precedence constraints. Equivalently, a solution is a bijection $f : \{1, \ldots, n\} \to V$ such that $(f(j), f(i)) \notin A$ for all $1 \leq i < j \leq n$, and its objective value –the number of setups– is $\sum_{i=1}^{n-1} s_i$, where $s_i = 1$ if $c_{f(i)} \neq c_{f(i+1)}$, and 0 otherwise. Clearly, no solution exists if there is a circuit in G. Hence, we assume that G has no circuit, which can efficiently be checked in a preprocessing step. An illustrative example of a PCCS instance is given in Figure 1. It consists of 17 operations, 3 classes (represented by colors white, gray and black), and 24 precedence constraints. The sequence $(9, 8, 12, 1, 2, 6, 10, 13, 3, 14, 4, 5, 11, 15, 17, 7, 16)$ of operations is an optimal solution with six setups.

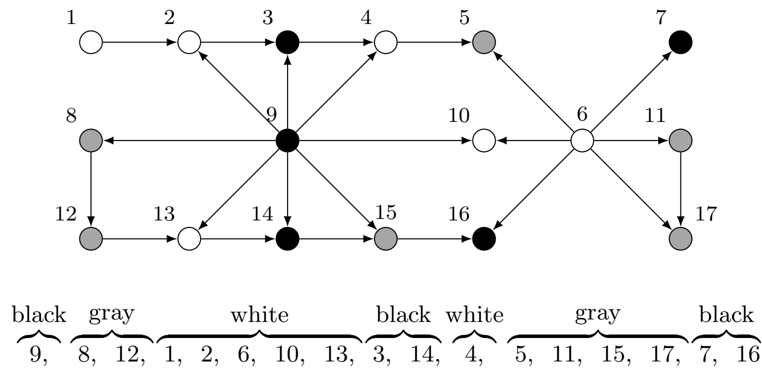


Figure 1: The precedence graph and an optimal sequence of a PCCS instance.

Lofgren (1986) and Lofgren et al. (1991) introduce the PCCS problem. The authors address the problem of routing printed circuit boards through an assembly cell, and they define the PCCS problem to capture this practical problem as a graph theoretic problem. They prove that the PCCS problem is NP-hard, show that two classes of heuristics have arbitrary bad worst case performance, develop a bunch of heuristics, and provide some computational results. In particular, the authors prove that the non-repetitive shortest common supersequence problem, which is a restricted but still NP-hard version of the well-known shortest common supersequence problem, is a special case of the PCCS problem. Altogether, their prolific contributions show that PCCS is a challenging and practically relevant optimization problem. This view is further supported by Tovey (2004) and Correa et al. (2007) who prove that no polynomial-time algorithm with constant worst-case performance exists for the PCCS problem unless $\mathcal{P} = \mathcal{NP}$. Note that the PCCS problem is trivial to solve if no precedence constraints exist or if there are at most two classes. However, as shown in (Lofgren et al., 1991; Darte, 2000), it becomes NP-hard in the strong sense if there are at least three classes, even if the precedence graph consists only of disjoint paths.

From a practical perspective, the PCCS problem models various recurring decision problems in systems where multipurpose processors, i.e., processors with the flexibility to perform multiple types of operations, need retooling or some other type of setup to switch from one operation to another (Tovey, 2004). Lofgren et al. (1991) describe an application in the semiconductor industry in detail. Darte (2000) mentions a traveling salesman application where the salesman (or another type of worker) must perform a set of tasks in some partial order, each task must be executed in some given city, and the goal is to minimize the number of moves

between cities. Camelot (2012) studies an aircraft disassembly process where reusable parts of an end-of-life aircraft have to be collected subject to precedence constraints, each part belongs to a specific zone of the aircraft, and the objective is to minimize the number of moves from one zone to another.

The PCCS problem is closely related to a loop fusion problem addressed by Darte (2000). Loop fusion is a standard compiler optimization and loop transformation process in computer science. The loop fusion problem of Darte (2000) can briefly be described as follows. Given is a set of loops, which is the same as the set of operations in the PCCS problem. Each loop belongs to some type (of loops) and a set of precedence-constraints between loops is specified. The goal is to find a total ordering of the loops so that the number of setups is minimized. A setup must be performed when switching from one type of loop to another, and –this is the difference with the PCCS problem– one can also specify the need for a setup between loops of the same type. These so-called fusion-preventing constraints make the loop fusion problem more general than the PCCS problem. Darte (2000) formulates the loop fusion problem as a graph theoretic problem, establishes a link to the scheduling literature, and develops complexity results for some variants of the loop fusion problem. We refer to his publication for further details about loop fusion.

The PCCS problem is also related to the following mixed graph coloring problem. Given is a mixed graph $G^{\text{mix}} = (V, A, E)$ where V is a set of vertices, A a set of arcs and E a set of edges. A function ϕ is a coloring of G^{mix} if it assigns to each vertex $v \in V$ a positive integer $\phi(v)$ such that $\phi(v) \leq \phi(w)$ if $(v, w) \in A$ and $\phi(v) \neq \phi(w)$ if $\{v, w\} \in E$. The number of colors of a coloring ϕ is given by the largest integer $\phi(v)$ assigned to a vertex $v \in V$. The goal is to find a coloring of G with a minimum number of colors. An instance $(V, A, \mathcal{C}, C)$ of the PCCS problem can be formulated as a mixed graph coloring problem as follows. The mixed graph $G^{\text{mix}} = (V, A, E)$ is obtained by simply adding an edge $\{v, w\} \in E$ to the precedence-graph $G = (V, A)$ for each pair of vertices v, w of distinct classes, i.e., $c_v \neq c_w$. Observe that the undirected part (V, E) of G^{mix} has a special structure: it is a complete k-partite graph where k equals the number $|\mathcal{C}|$ of classes. Given a coloring ϕ, a solution to the PCCS problem can be obtained as follows. According to coloring ϕ, we first execute all operations v with $\phi(v) = 1$, then all operations v with $\phi(v) = 2$, and continue this process until all operations are executed. Considering a group of operations with the same color, we execute them in an order that respects the precedence constraints A, which can easily be found as the precedence graph G has no circuit. The so-obtained total ordering of the operations is a solution to the PCCS problem, and the number of setups is exactly the number of colors of ϕ minus 1 as all operations with the same color $\phi(v)$ belong to the same class. As a result, a coloring of G^{mix} with a minimum number of colors corresponds to a solution of the PCCS problem with a minimum number of setups.

The discussed mixed graph coloring problem is introduced by Sotskov et al. (2002). The authors discuss some theoretical properties and devise an exact algorithm for solving this problem. A more effective exact algorithm is subsequently proposed by Andreev et al. (2000), who develop a branch-and-bound algorithm based on some conflict resolution and arc adding strategies. The authors state that their approach is less suited for mixed graphs where the number of edges is larger than the number of arcs as their lower bounding procedures are not taking into account the edges. In PCCS instances, the number of edges is typically much larger than the number of arcs. Hence, solving PCCS instances via their mixed-graph coloring approach seems to be unpromising. Meuwly et al. (2010) propose heuristic solution approaches for the mixed-graph coloring problem. Their tabu and variable neighborhood search methods are based on particol (see Blöchliger and Zufferey, 2008), which is a robust tabu search heuristic for the standard graph coloring problem. Additional theoretical investigations and algorithms for a slightly different mixed graph coloring problem are found in (Hansen et al., 1997; Ries, 2007; Kouider et al., 2017).

Computational experiments show that PCCS instances of practically relevant sizes cannot be solved to optimality using standard (mixed) integer linear programming approaches. Altogether, we therefore conclude that there exists no viable exact approach for solving the PCCS problem.

The remainder of this article is organized as follows. The next section proposes a branch-and-bound solution approach for the PCCS problem. After introducing the general breadth-first-search brand-and-bound structure, the specific sub-procedures incorporated into the branch-and-bound approach are discussed. Section 3 provides a computational study, which is based on 576 instances obtained from an instance generation scheme of Lofgren et al. (1991). The results of the branch-and-bound approach are used to discuss the diffi-

culty of the generated instances and compared to the results of two integer linear programming approaches. Concluding remarks are given in Section 4. Detailed computational results are provided in the appendix.

2 A branch-and-bound algorithm for the PCCS problem

Given a PCCS instance $(V, A, \mathcal{C}, C)$, a solution can be constructed by successively selecting a next class and executing all possible operations of this class so that the precedence constraints are satisfied, until all operations are executed. This scheme, introduced by Lofgren et al. (1991), gives an indirect representation of the solutions by class sequences, which makes it possible to specify the optimization problem on the solution space of the class sequences.

This can be formalized as follows. Let $\widehat{G} = (V, \widehat{A})$ be the transitive closure of precedence graph $G = (V, A)$, i.e., an arc (v, w) belongs to $\widehat{A}$ if and only if there exists a path from v to w in G. A vertex (or operation) v is *available* if there is no w in G with $(w, v) \in \widehat{A}$ and $c_w \neq c_v$, and a class $c \in \mathcal{C}$ is *available* if it contains at least one available vertex. We define the *execution* of an available class $c \in \mathcal{C}$ to be the deletion of all available vertices $v \in V$ of class $c_v = c$.

A class sequence $S = (c(1), c(2), ...c(r))$ is *feasible* if every $c(i)$, $i \in \{1, \ldots, r\}$, is available after the execution of $c(1), \ldots, c(i-1)$, and the successive execution of the classes in S results in the deletion of all vertices of G. Given any feasible class sequence S, a solution to the PCCS problem, which is a total ordering of the operations, can simply be obtained by additionally sequencing all operations that are deleted at the same time in S. This is a simple task as G has no circuit, and the objective value of the resulting solution (i.e., the number of setups) is the length of the class sequence S minus one. We call a feasible class sequence *optimal* if it is of shortest length among all feasible class sequences.

Clearly, not all solutions to the PCCS problem are representable by a feasible class sequence. However, when executing an operation, it is profitable to perform all available operations of the same class before switching to another class. Consequently, there exists an optimal solution to the PCCS problem that is representable by a feasible class sequence, and this is an optimal class sequence.

In order to find an optimal class sequence, we develop a breadth-first-search branch-and-bound algorithm that iteratively constructs feasible class sequences and branches on the next available classes. For a general understanding of branch-and-bound algorithms, we refer the reader to standard textbooks of the field such as (Papadimitriou and Steiglitz, 1998). To avoid confusion with vertices of G, vertices of the branch-and-bound tree are called *nodes*. The algorithm, whose skeleton is provided in Algorithm 1, can be described as follows.

As first step, we generate an initial branch-and-bound node consisting of the precedence graph $G = (V, A)$, the set $\mathcal{C}$ of classes, and the class vector C. We further add an initial empty sequence S that keeps track of the executed classes. Variables *bestSolution* and *lowerBound* refer to the best class sequence found so far and to the current lower bound, respectively.

Before starting the actual branch-and-bound process, we try to simplify the input by sequentially applying specialized procedures called *merging*, *heuristic*, *reasoning*, *bounding* and *immediate selection*, until the node is structurally not changed anymore (lines 2 to 8). The *merging* procedure tries to merge same-class operations in precedence graph $G = (V, A)$, the *heuristic* procedure tries to find a new best feasible class sequence by extending the partial class sequence provided by the node, the *reasoning* procedure tries to add implicit precedence constraints, the *bounding* procedure gives a lower bound for the node, and the *immediate selection* procedure executes an available class if it can be identified as optimal next choice. The applied procedures are described in detail in the next subsections.

If the lower bound obtained in the preprocessing step is not strictly smaller than the objective value of *bestSolution*, an optimal sequence is found and we can stop the search. Otherwise, we start the branch-and-bound process in a breadth-first-search manner. List *currentNodes* contains all nodes of the current tree level.

Algorithm 1: Breadth-first-search branch-and-bound algorithm

```
1  Generate an initial node with the input (V, A, 𝒞, C), add an empty sequence S to this node, set bestSolution to empty
     and lowerBound to 0;
2  repeat
3      apply merging to node;
4      apply heuristic to node and update bestSolution;
5      apply reasoning to node;
6      apply bounding to node and update lowerBound;
7      apply immediate selection to node and update its sequence S;
8  until node did not change;
9  set currentNodes to empty;
10 if lowerBound is strictly smaller than the objective value of bestSolution then
11     add node to currentNodes;
12 while currentNodes is not empty do
13     set nextLevelNodes to empty;
14     foreach node in currentNodes do
15         apply immediate selection to node and update its sequence S;
16         if a class was executed in this procedure then
17             add node to nextLevelNodes;
18         else
19             foreach available class of node do
20                 copy the current node;
21                 execute the class in the copied node and update its sequence S;
22                 add the resulting node to nextLevelNodes;
23             end
24     end
25     delete duplicated and dominated nodes in nextLevelNodes;
26     foreach node in nextLevelNodes do
27         apply heuristic to node and update bestSolution;
28     end
29     set lowerBound to objective value of bestSolution;
30     foreach node in nextLevelNodes do
31         apply merging and reasoning to node;
32         apply bounding to node, obtaining lowerBoundOfNode;
33         if lowerBoundOfNode is not strictly smaller than the objective value of bestSolution then
34             delete node from nextLevelNodes;
35         else
36             if lowerBoundOfNode is strictly smaller than lowerBound then
37                 set lowerBound to lowerBoundOfNode;
38     end
39     set currentNodes to nextLevelNodes;
40 end
```

The next level branch-and-bound nodes are generated as follows (lines 13 to 39). We first initialize list *nextLevelNodes* to empty. For each node in *currentNodes*, we try to find an optimal next class with the *immediate selection* procedure. If we find one, we execute this class and add the node to the list *nextLevelNodes* (lines 15 to 17). Else, we branch on the available classes (lines 19 to 23). For each available class, we copy the current node, execute the class in the copied node, update sequence S and add the resulting node to list *nextLevelNodes*.

In order to reduce the number of nodes, we search for duplicated and dominated nodes in the list *nextLevelNodes* (line 25) as follows. If the sets of the non-executed operations of two nodes are the same, we arbitrarily delete one from *nextLevelNodes*. Clearly, the list of the executed classes of the two nodes are different, but the length of these lists is the same as the branch-and-bound approach works in breadth-first manner. This makes it possible to discard one of the two nodes in the search. Similarly, if the set of non-executed operations of some node a in *nextLevelNodes* is strictly included in the set of non-executed operations of some other node b *nextLevelNodes*, then we can discard node b as a obviously dominates b.

In order to possibly find a new best solution, we then apply the *heuristic* procedure to each node in list *nextLevelNodes* and update *bestSolution* if some new best sequence is found (lines 26 to 27). For each node in list *nextLevelNodes*, after applying the *merging* and *reasoning* procedures (line 31), we compute a lower bound for the node with the *bounding* procedure (line 32). If the obtained bound is not smaller than

the objective value of the best solution found so far, we delete the node from list $nextLevelNodes$ (lines 33 to 34) as no extension of sequence S can be shorter than the best sequence found so far. The overall lower bound $lowerBound$ is then set to the lowest lower bound of all nodes kept in list $nextLevelNodes$ (lines 47 to 48). Finally, we update the list $currentNodes$ to $nextLevelNodes$ (line 39).

The search stops if the list $currentNodes$ becomes empty (line 12), which means that no more nodes need to be explored and $bestSolution$ contains an optimal feasible class sequence.

The crucial components of the branch-and-bound approach are the specialized *merging*, *heuristic*, *reasoning*, *bounding*, and *immediate selection* procedures, which are applied to the branch-and-bound nodes. We now describe these procedures in detail.

2.1 Merging

Given a branch-and-bound node with its executed class sequence S and its precedence graph $G = (V, A)$ that contains only those vertices that were not removed by the sequential execution of the classes in S, we try to find same-class operations that can always be executed at the same time in a class sequence. This will make it possible to merge these operations in G.

For each operation $v \in V$, denote by $Pred(v) = \{w \in V : c_w \neq c_v \text{ and } (w, v) \in \widehat{A}\}$ the set of different-class predecessors of v in graph G, recalling that $\widehat{G} = (V, \widehat{A})$ is the transitive closure of G. If, for two distinct same-class vertices v and w, i.e., $c_v = c_w$, the sets $Pred(v)$ and $Pred(w)$ contain exactly the same vertices, then the two operations always become available for execution at the same time. Hence, if we choose to execute class c_v at some point in time, we either execute both operations v and w or none of them. Therefore, we can represent the two operations v and w by a single operation q in $G = (V, A)$ as follows. First, we add a vertex q to V. Then, for each vertex $p \in V \setminus \{v, w\}$, we add an arc (p, q) to A if $(p, v) \in A$ or $(p, w) \in A$, and an arc (q, p) if $(v, p) \in A$ or $(w, p) \in A$. Finally, we delete both v and w and all their incident arcs. The class c_q associated to operation q is $c_v = c_w$, and we keep track of this merging operation by storing the mapping of q to set $\{v, w\}$ in order to reconstruct the solution for the original instance at the end.

Considering our illustrative example of Figure 1, we observe that both white vertices 1 and 6 have no predecessors. Hence, they can be merged to a single vertex called $1; 6$. The resulting graph is depicted in Figure 2 a). In this graph, we observe that vertices 2 and 10 can be merged as they have the same different-class predecessors, namely vertex 9. After this merging, we obtain the graph depicted in b). Similarly, we obtain c) after merging vertices 11 and 17 and d) after merging vertices 8 and 12.

A second type of merging opportunity arises if the sets of different-class predecessors and successors of a vertex include the corresponding sets of another same-class vertex. More formally, for each vertex $v \in V$, denote by $Succ(v) = \{w \in V : c_w \neq c_v \text{ and } (v, w) \in \widehat{A}\}$ the set of different-class successors of v in G. If, for two distinct same-class vertices v and w, $Pred(v) \subseteq Pred(w)$ and $Succ(v) \subseteq Succ(w)$, then the two operations v and w can be merged in the same way as discussed before. The reasoning is the following. Consider any feasible class sequence $S' = (c(1), \ldots, c(r))$ that removes all non-executed operations V of this branch-and-bound node. Let i_v and i_w be such that v is removed by the execution of $c(i_v)$, and w is removed by the execution of $c(i_w)$. As $Pred(v) \subseteq Pred(w)$ holds, we have $i_v \leq i_w$. Suppose $i_v < i_w$ and let W be the set of successors of v of class c_v removed by the execution of a $c(k)$ with $k < i_w$. Also, let j be such that $i_v < j < i_w$ and $c(j) \neq c(i_v) = c(i_w)$. As $Succ(v) \subseteq Succ(w)$, there exists no path in G from a vertex in $\{v\} \cup W$ to any vertex removed by the execution of $c(j)$. The removal of the vertices in $\{v\} \cup W$ can therefore be delayed and performed together with the removal of w. The so-created class sequence S'' is feasible and its length is not larger than the length of S'.

This type of merging is illustrated in our illustrative example of Figure 1 from d) to e), where operations 7 and 16 are merged. The only predecessor of vertex 7 in d) is vertex 1;6 which is one of the two predecessors of vertex 16, and both vertices 7 and 16 have no successors. Hence, we can merge these two vertices as $Pred(7) \subset Pred(16)$ and $Succ(7) = Succ(16)$ holds. Similarly, we observe in e) that vertices 5 and 11;17 can be merged. The resulting graph is depicted in f). The *merging* procedure is then stopped as there are no more detected merging opportunities.

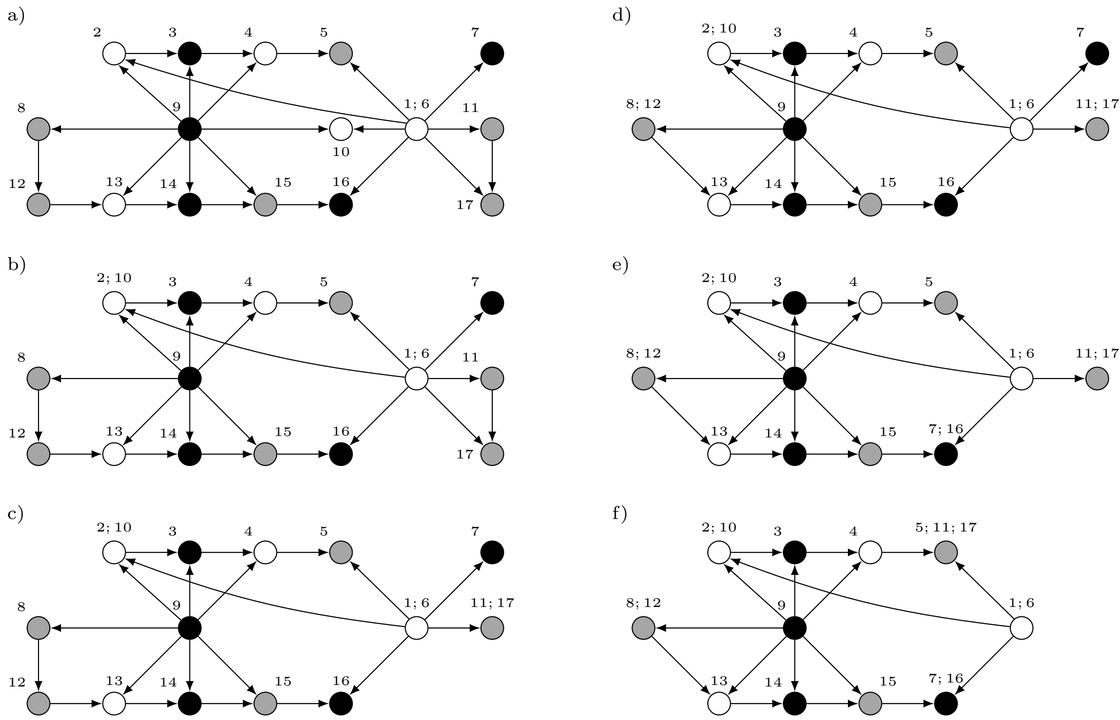


Figure 2: Six successive merging steps in our illustrative example.

2.2 Bounding

Consider again a branch-and-bound node with its partial sequence S of executed classes and its precedence graph $G = (V, A)$ that contains only those vertices that were not removed by the sequential execution of the classes in S. We now compute a lower bound on the length of a class sequence that extends S to a feasible one. Such a bound can easily be obtained by the following critical-path-based calculation. Let $G^+ = (V^+, A^+, d)$ be the graph obtained from $G = (V, A)$ by adding a source vertex σ, a sink vertex τ (i.e., $V^+ = V \cup \{\sigma, \tau\}$), an arc (σ, v) with length 1 and an arc (v, τ) with length 0 for all $v \in V$ (i.e., $A^+ = A \cup \{(\sigma, v), (v, \tau) : v \in V\}$). For each arc $(v, w) \in A$, we define its length d_{vw} as 1 if $c_v \neq c_w$, and 0 otherwise. The length d_{vw} reflects a class change between the execution of v and w, and the length of 1 for all arcs incident to the source accounts for the execution of the first class. It is then easy to see that the length of any path from σ to τ is a lower bound on the number of remaining class executions to perform all operations of V. Let $LP(G^+)$ be the length of a longest path from σ to τ in G^+. If c is the last class of S, then no vertex of class c is available in G since they have all been removed by the execution of c. Hence, given the partial class sequence S of length $|S|$, a lower bound on the length of a feasible class sequence extending S is $|S| + LP(G^+)$. This type of lower bound computation is also proposed by Lofgren et al. (1991).

Consider again the graph depicted in Figure 2 f), which was obtained by applying the *merging* procedure to our illustrative example. The critical-path-based calculation gives $\{\sigma, 9, 8;12, 13, 14, 15, 7;16, \tau\}$ as vertex set of a longest path form σ to τ in G^+. Since $|S| = 0$ and $LP(G^+) = 6$, the lower bound is 6.

A stronger lower bound can be obtained by determining the minimum number r_c of remaining class executions needed to remove all vertices of each class $c \in \mathcal{C}$. The *one-class-based lower bound* is then defined as $|S| + \sum_{c \in \mathcal{C}} r_c$. Each value r_c is calculated in graph $G_c^+ = (V^+, A^+, d^c)$ obtained from G^+ by changing the arc lengths as follows. For each arc $(v, w) \in A$, we set d_{vw}^c to 1 if $c_v = c$ and $c_w \neq c$, and 0, otherwise. Also, for each vertex $v \in V$, we set $d_{\sigma v}^c$ to 1 if $c_v = c$ and 0 otherwise. Then, the length of any path from σ to τ is a lower bound on the number of remaining class executions for class c. Hence, r_c is the length of a longest path from σ to τ in G_c^+. For the graph depicted in Figure 2 f), we obtain a one-class-based lower bound of 7. Indeed, we get a lower bound of 2 for the white vertices, 2 for the gray ones, and 3 for the black ones. As $|S| = 0$, the overall lower bound is $2 + 2 + 3 = 7$.

It can easily be proved that the one-class-based lower bound is at least as large as $|S| + LP(G^+)$. Hence, we decided to use only the one-class-based lower bound in our implementation.

2.3 Reasoning

Given a branch-and-bound node with its partial sequence S of executed classes, and given an upper bound UB on the length of an optimal feasible class sequence, we try to find additional precedence-constraints that must be satisfied by any feasible class sequence extending S that is of length at most $UB-1$. Such constraints can be added as we already have a solution with objective value UB at hand, so that we can restrict the search space to all solutions with objective values at most $UB-1$.

So, let $G = (V, A)$ be the precedence graph that contains only those vertices that were not removed by the sequential execution of the classes in S. For each pair v, w of distinct vertices in G such that $(v, w) \notin \widehat{A}$ and $(w, v) \notin \widehat{A}$, we check with the *bounding* procedure if we can obtain a lower bound of at least UB after adding precedence constraint (v, w) to A. If the answer is yes, w is executed not later than v in any feasible class sequence with length at most $UB-1$. Therefore, (w, v) can be added to A, which forbids to perform w later than v. Clearly, operations v and w are still allowed to be performed at the same time if both are from the same class. Similarly, if we can obtain a lower bound of at least UB after adding precedence constraint (w, v) to A, then we add (v, w) to A to avoid w being removed before v. Once no more arcs can be added to A, we update G to its transitive reduction (see, e.g., Aho et al., 1972). This helps to reduce the computational effort when determining critical paths in G.

Consider again the graph depicted in Figure 2 f), and assume we already found a feasible class sequence of length 8, so that $UB = 8$. We check if we can add the precedence-constraint (1;6, 14). For this purpose we temporarily add the opposite arc (14, 1;6) to A and calculate the one-class-based lower bound. We get 3 for all three classes, so that the overall lower bound is 9. As this is not lower than UB, we add (1;6, 14) to A. The *reasoning* procedure then sequentially adds arcs (2;10, 14), (3, 15), (4, 15), (5;11;17, 7;16), (8;12, 3), (9, 1;6), (13, 3), (14, 4), and (8;12, 1;6) to A, obtaining the graph depicted in the left part of Figure 3. After applying the transitive reduction, we finally get the graph depicted in the right part of this figure.

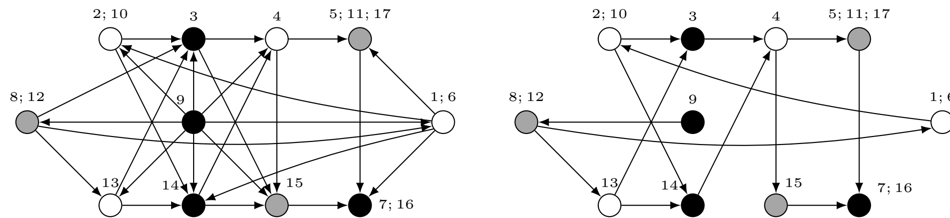


Figure 3: Output of the *reasoning* procedure when applied to the graph of Figure 2 f).

2.4 Immediate selection

Once more, consider a branch-and-bound node with its partial sequence S of executed classes and its precedence graph $G = (V, A)$ that contains only those vertices that were not removed by the sequential execution of the classes in S. We check if we can find a next class to be executed, which is a class that is performed first in at least one optimal feasible class sequence for the set of non-executed operations V.

For this purpose, we first determine the set of available classes. If there is only one available class c, then c is clearly an optimal next choice. Otherwise, we look for an available class c that has no possibility to be extended before its execution since, in such a case, it makes no sense to further wait for the execution of c. More formally, let V_c^{av} be the set of available operations of a an available class c, and let $V_c^{\text{nav}} = \{v \in V : c_v = c\} \setminus V_c^{\text{av}}$ be the set of non-available operations of class c. Consider a vertex $u \in V_c^{\text{nav}}$. If there is a path P from a vertex $v \in V_c^{\text{av}}$ to u in G such that at least one vertex on P, say p, is of class $c_p \neq c$, then u cannot be removed together with the vertices of V_c^{av} in a class execution. Indeed, as there is a path from v to p and $c_p \neq c_v$, p will be removed after v and, similarly, u will be removed after p in any feasible class sequence. Hence, u will never be available when v will be removed together with all vertices of V_c^{av} (and possibly other

vertices). Now, if we can find such a path for all operations u in V_c^{nav}, this means that we will never be able to extend the current set of available operations for c, and we can therefore execute class c right away.

Consider again our illustrative example. After applying the reasoning procedure, we obtain the graph depicted in Figure 3. Only the black class is available, and it contains only one available operation, namely vertex 9. Note that this class cannot be extended since the paths $(9, 8;12, 13, 3)$, $(9, 8;12, 13, 14)$ and $(9, 8;12, 13, 14, 4, 15, 7;16)$ connect vertex 9 to the three other black vertices 3, 14 and 7;16, respectively, and they all contains the gray vertex 8;12. Hence, there are two reasons for which we can execute the black class by performing operation 9 : it is the only available class, and it is not extendable. After deleting vertex 9 from G, we observe that the gray class can be executed as it is the only available class. Subsequently, our *immediate selection* procedure is able to execute the class sequence white, black, white, gray, black, obtaining an optimal class sequence with length 7 for the given instance.

2.5 Heuristics

In a branch-and-bound approach, it is also important to determine good upper bounds in order to discard nodes (see lines 10–11 and 33–34 in Algorithm 1) and to successfully apply the *reasoning* procedure. This is the task of the *heuristic* procedure, which works as follows.

Given a branch-and-bound node with its partial sequence S of executed classes and an upper bound UB on the length of an optimal feasible class sequence, we try to extend S to a feasible class sequence of length at most $UB - 1$. For this purpose, we apply the *merging* procedure and then try to find a next class by the *immediate selection* procedure. If this procedure does not find an optimal next class, we determine the next class by a heuristic *rule*. We restart these steps until V contains no more operations. Algorithm 2 contains a pseudo-code of this generic constructive heuristic.

Algorithm 2: A generic heuristic for the PCCS problem

```
1 Set S' equal to S;
2 while S' is not feasible and |S'| < UB do
3 |   apply merging to node;
4 |   apply immediate selection to node and update S';
5 |   if no class was executed in this immediate selection procedure then
6 |   |   select an available class with some rule, execute it, and update S';
7 end
8 if |S'| < UB then return S';
9 else return null;
```

Clearly, the performance of this heuristic is mainly determined by the *rule* (line 6) that selects a class among all available ones. We consider the following rules, which are introduced by Lofgren et al. (1991):

Random Randomly choose an available class.

Greedy Choose an available class with a largest number of available operations. We hereby count the number of available operation with respect to the original problem. If, for example, some operation obtained through merging steps is available, we count the number of operations it corresponds to in the original instance.

Altruistic Choose an available class that would, if executed, make the largest number of new operations available. Again, we count the number of newly available operations with respect to the original instance.

Critical path level For each available operation $v \in V$, determine the length l_v of a longest path from v to τ in G^+ (see Section 2.2 for the details). An available vertex v is called critical if l_v is equal to $LP(G^+) - 1$, and a class is called critical if it contains at least one critical vertex. We then choose a critical class. Hereby, we apply one of the three rules described before, i.e., random, greedy, or altruistic, for breaking ties, which gives three different heuristics for this rule.

For the input graph of Figure 1 and a (unrestricted) upper bound of $UB = |V| + 1$, the heuristic finds the same optimal sequence that we found at the end of Section 2.4 with all rules, except with the greedy rule, which finds a class sequence of length 8.

3 Computational results

In order to evaluate the performance of the developed branch-and-bound algorithm, we test it on benchmark instances proposed by Lofgren et al. (1991) for routing printed circuit boards through an assembly cell. The approach is implemented in Java and run on a PC with an Intel Core i7-6700 3.4 GHz processor and 32 GB memory. In this section, we first describe the instance generation scheme, evaluate then the *heuristic* procedures, and finally analyze the performance of the branch-and-bound approach.

3.1 Instance generation

The application and the benchmark instances we consider can briefly be described as follows. We refer to Lofgren et al. (1991) for the details. Given are a flexible assembly cell with W workstations and a rectangular board of size (X, Y) where components are placed at the integer points of an X times Y grid. At each workstation, a given set of components are available, and each component is present at exactly one workstation. In the PCCS problem, each workstation is represented by a class, each component on the board is an operation, and the class of an operation is given by the workstation where the component is available. The goal is to determine a complete ordering of the component assembly operations so that the number of times the board has to be moved from one workstation to another is minimized, which is the same as minimizing the number of setups in the PCCS problem.

There are three different sizes of components: large, medium, and small. Parameter L refers to the number of large components. Any non-large component is either small with a probability P, or medium otherwise. The small and medium components occupy only one grid point while large components take three consecutive grid points, either vertically or horizontally. Every grid point is used by exactly one component. Any component can be placed at any location, except that two large components cannot be adjacent to each other.

Two types of instances are considered. In *dense* instances, each large component has precedence over all components around its perimeter (i.e., left, right, above, and below), each medium component has precedence over any medium or small component to its right and over any medium component below it, and each small component has precedence over any medium or small component to its right. In *non-dense* instances, we have the same precedence constraints as in the dense ones, except that the small components have no precedence over the small ones to their right.

Inspired by Lofgren et al. (1991), the parameters are set as follows in our benchmark set. We consider small boards of size $(20, 10)$ and large boards of size $(30, 15)$. The number of workstations W belongs to $\{3, 4, 5, 7\}$ and the components are randomly assigned to the workstations. The number L of large components is in $\{0, 4, 8, 12\}$ and the probability P in $\{0.5, 0.7, 0.9\}$. Both dense and non-dense instances are considered. For every parameter setting, we randomly generate three instances, which gives a total of $4 \times 4 \times 3 \times 2 \times 3 = 288$ small instances with board size $(20, 10)$, and 288 large instances with board size $(30, 15)$. For illustration purpose, we depict a small dense instance in Figure 4, while a small non-dense instance is shown in Figure 5.

3.2 Evaluation of the heuristics

The six different versions of the general heuristic are evaluated in order to see whether some versions are better suited to be included in the branch-and-bound algorithm. For this purpose, we first run each version on all 576 benchmark instances (using an unrestricted upper bound) and then, for each version and instance, we compute the relative percent deviation (RPD) of the attained objective value (res) to the lowest objective value ($best$) over all versions, i.e., RPD $= 100(res - best)/best$. Figure 6 illustrates the RPDs by showing

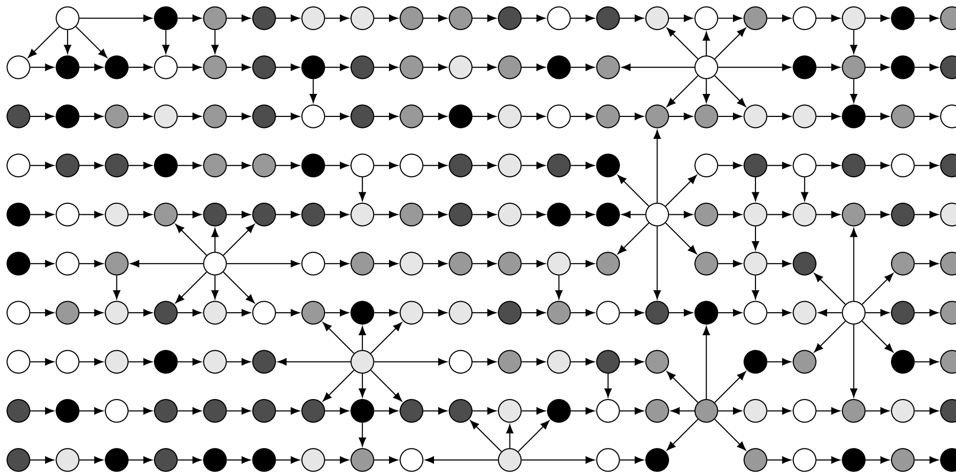


Figure 4: A small dense instance with $W = 5$, $L = 8$ and $P = 0.7$. Each workstation is represented by a color, and each component is depicted by a vertex having the color of the workstation to which it is assigned. Each arc represents a precedence relation.

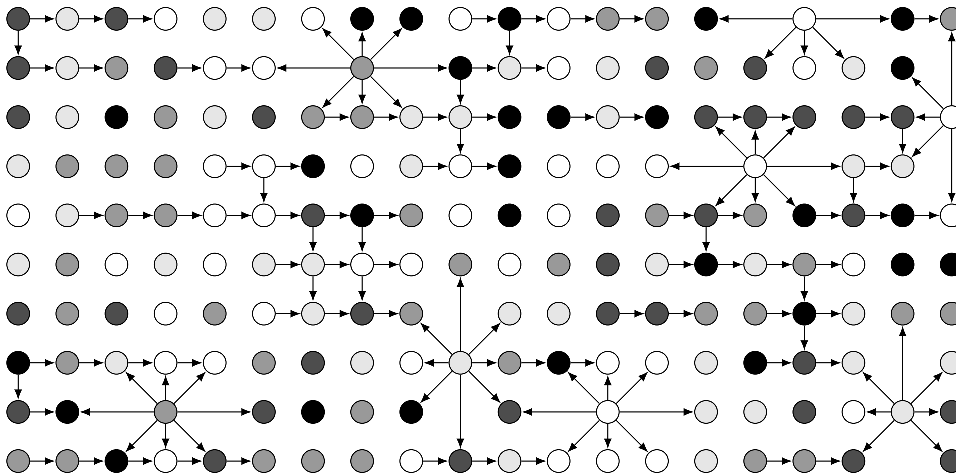


Figure 5: A small non-dense instance with $W = 5$, $L = 8$ and $P = 0.7$.

the minimum, first quartile, median, third quartile, and maximum RPDs in box-and-whisker plots, using standard conventions for drawing such plots. Additionally, the average RPDs are represented by diamonds. The following can be observed.

As expected, the random rule, which is mainly introduced for comparison purposes, is outperformed by all other rules: its RPDs are substantially higher than those of the other rules. The performance of the greedy and altruistic rules are similar (i.e., their box plots have a similar shape). A substantially better performance is obtained with the three critical-path-based rules. The average RPDs are, for example, 5.2%, 3.1%, and 2.5% for the CP-rule with tie-breaking strategy random, greedy, and altruistic, respectively. Furthermore, CP-greedy and CP-altruistic find a best solution in more than 50% of the instances, which is shown by a median RPD of 0%. As expected, these two versions perform slightly better than CP-random.

While the CP-random, CP-greedy, and CP-altruistic heuristics find a best solution in 204, 327, and 345 instances, respectively, at least one of the three approaches finds a best solution in 525 instances. These numbers suggest that running all three versions is a good strategy. We therefore decide to run all three versions when applying the *heuristic* procedure in the branch-and-bound algorithm.

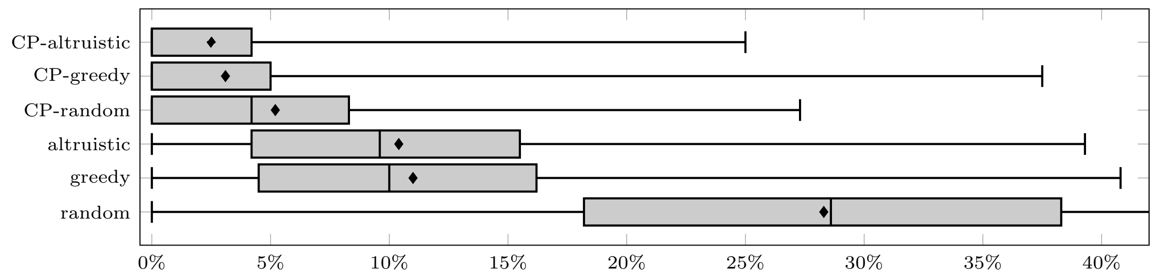


Figure 6: Box plots of the RPDs attained with the different versions of the generic heuristic.

3.3 Evaluation of the branch-and-bound approach

In order to analyze the performance of the branch-and-bound approach, we execute a run for each benchmark instance using a time limit of 7200 s per run. If the time limit is exceeded, we stop the search and record the best solution found and the current lower bound. Detailed results are given in Tables 1 and 2 of the appendix.

As first comment, let us mention that the branch-and-bound approach solves 275 of the 288 small instances and 175 of the 288 large instances to optimality. Interestingly, and at first unexpectedly for us, most of the small instances and a considerable number of large instances are solved within a few seconds. Indeed, the search finishes within 20 s for 200 small and 101 large instances.

3.3.1 Easy and hard instances

As the solution times vary considerably from one instance to another, it is of interest to analyze which instances are easy and which are hard to solve by the branch-and-bound algorithm. This gives an idea about the difficulty of the instances in general. For this purpose, we analyze the solution times and check the parameter values of the unsolved instances, i.e., those which could not be solved to optimality.

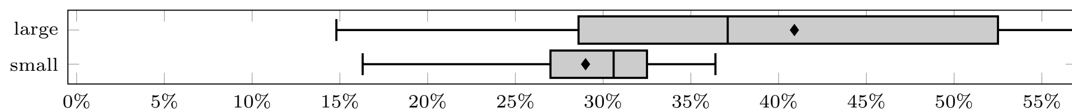
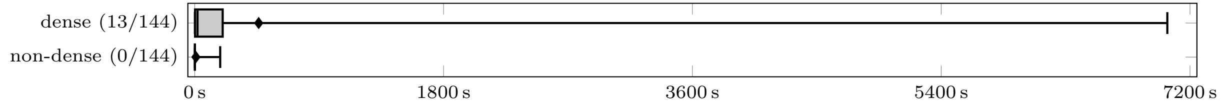


Figure 7: Box plot of the relative optimality gaps for the unsolved small and large instances.

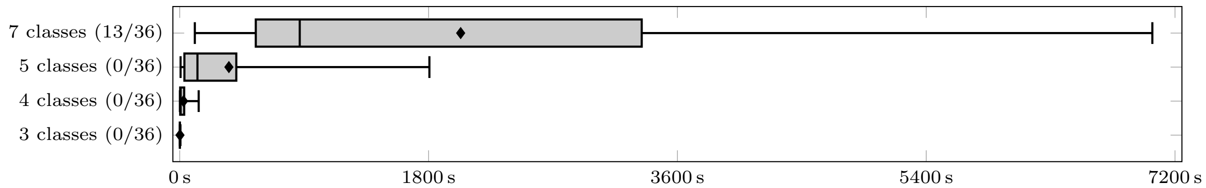
First of all, we note that the size of the board has a big impact on the performance of the algorithm: the larger the board, the more operations are to be sequenced, and the more difficult the instances tend to be. This view is supported by the portion of solved instances, which is about 95% (275 out of 288) and 61% (175 out of 288) for the small and large instances, respectively. Furthermore, as depicted in Figure 7, the relative optimality gaps of the unsolved instances are substantially higher for the large instances.

The density also has an important impact on the difficulty of an instance, as illustrated in Figure 8, which shows the run times in a box plot for all small instances grouped by density. The figure also provides the number of unsolved instances and the total number of instances for each group in brackets. We note that 13 out of 144 small dense instances could not be solved to optimality, while all non-dense instances could be solved within three minutes. This can be explained as follows. If an instance is dense, each component has precedence over or is preceded by the component to its right, see e.g., Figure 4. Hence, all vertices in a same row of the board are weakly-connected. As a result, the number of weakly-connected components of the precedence graph is bounded by the number Y of rows of the board, and the size of a weakly-connected component of the graph is at least the number X of columns of the board minus 2 times the number of large components of the given row. This is quite large. For comparison, the small components of a non-dense instance have no precedence over the small component to their right. As a result, the precedence graph of non-dense instances typically contains many very small weakly-connected components such as isolated vertices. This is illustrated in Figure 5, which shows a non-dense instance. The *merging* procedure is then able to merge a significant number of vertices in non-dense instances as it is somewhat easy to merge small

i) Solution times for the small instances, grouped by density



ii) Solution times for the small, dense instances, grouped by number of classes



iii) Solution times for the small, dense instances with 7 classes, grouped by number of large components (above) and small component probability (below)

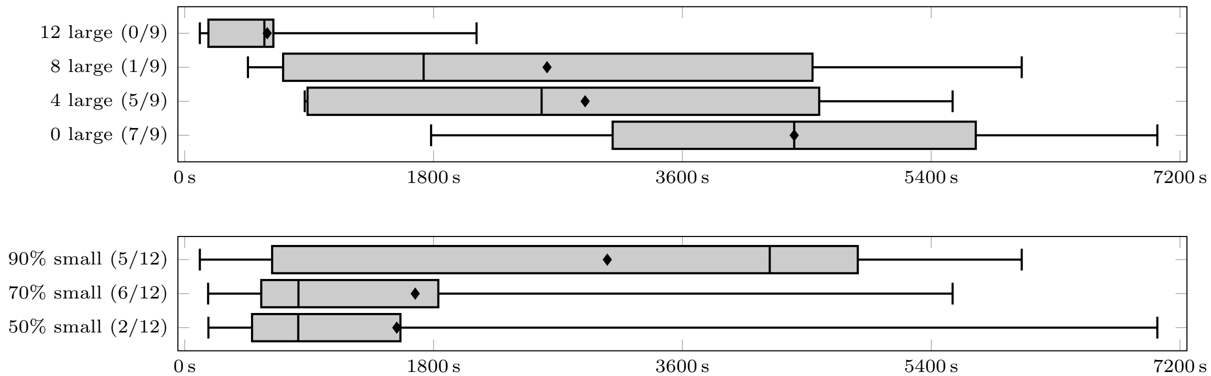


Figure 8: Box plots of the run times attained for some subsets of instances. In brackets, the number of unsolved instances and the total number of instances for each subset.

weakly-connected components with other vertices. For example, the 184 vertices of the instance of Figure 5 are replaced by 37 vertices after only one application of the *merging* procedure. Clearly, this reduction of the number of operations simplifies the solution process significantly.

As the difficulty to solve dense instances varies quite a lot from one instance to another, we further analyze them. As shown in Figure 8 ii), the number W of classes is also an important factor for the difficulty. Instances with less than five classes are solved almost instantaneously, while it takes about 350 s, on average, to solve an instance with five classes. As expected, the instances with seven classes are the most challenging ones. All 13 unsolved instances contain seven classes, and the run times of the solved instances are significantly higher than those of the instances with less than seven classes. This finding is no surprise: the more classes involved, the more alternatives one typically has in choosing the next class to execute.

When further analyzing the 36 dense instances with seven classes, one can see in Figure 8 iii) that instances with many large components seem to be easier than the others. This can be explained as follows. Instances with many large components have a smaller total number of components, and since such components have priority over all neighboring components, they must be scheduled quite early, which helps to simplify the search.

Similarly, Figure 8 iii) also shows that instances with a small number of small components tend to be easier than the other instances. On dense instances, the only difference between small and medium components is the precedence of medium components over any medium component directly below them. These additional

precedence constraints seem to simplify the search. Indeed, among the 13 unsolved small instances, we find all instances with $W = 7$ classes, $L = 0$ and $P = 90\%$, which means that about 90% of the components are small and, only 10% are of medium size. As a result, the precedence graph of these instances consists of 20 paths of length 10 with almost no precedence constraints between vertices of different paths. Similar instances are used in (Lofgren et al., 1991; Correa et al., 2007) to establish various NP-hardness results for the PCCS problem. Hence, our computational analysis supports the view that this type of PCCS instances are difficult to solve.

3.3.2 Comparison of the branch-and-bound algorithm with two integer linear programs

As no comprehensive computational results are available for the PCCS problem, we decide to compare our branch-and-bound algorithm with two integer linear programming formulations of the PCCS problem. The first formulation is similar to standard continuous-time formulations of machine scheduling problems with sequence-dependent setup times. For each pair $(i, j) \in V \times V$ of distinct operations, introduce a binary variable y_{ij}, which takes value 1 if i is executed before j, and 0 otherwise. In order to count the number of setups, introduce an integer variable x_i for each operation $i \in V \cup \{\tau\}$ reflecting the number of setups necessary before starting the execution of operation i. Hereby, τ is a fictive end operation executed after all other operations. For each pair of operations $i, j \in V$, parameter s_{ij} indicates if a setup is needed when executing j directly after i. Therefore, set $s_{ij} = 1$ if $c_i \neq c_j$, and 0 otherwise. Also, let U be an upper bound on the optimal total number of setups. In our computational tests, we set U to the objective value of a solution obtained by the CP-greedy heuristic (see Section 2.5). The PCCS problem can then be described by the following integer linear program, called IP1:

$$
\begin{aligned}
&\text{Minimize } x_\tau && && \text{(1a)}\\
&\text{subject to} && && \\
&\quad y_{ij} + y_{ji} = 1 && \text{for all distinct } i, j \in V && \text{(1b)}\\
&\quad x_j - x_i - (s_{ij} + U)y_{ij} \geq -U && \text{for all distinct } i, j \in V && \text{(1c)}\\
&\quad x_i - x_j - (s_{ji} + U)y_{ji} \geq -U && \text{for all distinct } i, j \in V && \text{(1d)}\\
&\quad y_{ij} = 1 && \text{for all } (i, j) \in A && \text{(1e)}\\
&\quad x_\tau \leq U && && \text{(1f)}\\
&\quad x_\tau \geq x_i && \text{for all } i \in V && \text{(1g)}\\
&\quad y_{ij} \in \{0, 1\} && \text{for all distinct } i, j \in V && \text{(1h)}\\
&\quad x_i \in \mathbb{Z}_{\geq 0} && \text{for all } i \in V \cup \{\tau\}. && \text{(1i)}
\end{aligned}
$$

Constraints (1b) ensure that all pairs of operations are sequenced. Constraints (1c) and (1d) link the sequencing variables y with the setup counting variables x. Indeed, if i is executed before j, constraints (1c) impose $x_j \geq x_i + s_{ij}$ and constraints (1d) impose $x_j \leq x_i + U$ (which is not restrictive), while $x_i \geq x_j + s_{ji}$ and $x_i \leq x_j + U$ are imposed if j is executed before i. Constraints (1e) specify that the precedences given in set A must be met. Constraints (1f) and (1g) make sure that the end operation τ is executed after all other operations and that there are at most U setups. Constraints (1h) tell that the variables y are binary and constraints (1i) specify that the variables x are non-negative integers. Clearly, x could also be specified as continuous variables. Finally, the objective is to minimize the total number of setups, which is reflected in (1a).

The second formulation is similar to typical discrete-time formulations of machine scheduling problems. For each operation $i \in V$ and each $p \in \{1, \ldots, U\}$, introduce a variable x_{ip}, which takes value 1 if there are exactly p setups performed before executing operation i, and 0 otherwise. Hereby, U is the same upper bound on the total number of setups as the bound used in the previous formulation. Introduce a variable z reflecting the total number of setups. Then the problem can be described by the following integer linear program, called IP2:

$$
\begin{aligned}
&\text{Minimize } z && && \text{(2a)}\\
&\text{subject to} && && \\
&\quad \sum_{p=1}^{U} x_{ip} = 1 && \text{for all } i \in V && \text{(2b)}\\
&\quad x_{ip} + x_{jp} \leq 1 && \text{for all } p \in \{1,\dots,U\} \text{ and } i,j \in V \text{ with } c_i \neq c_j && \text{(2c)}\\
&\quad x_{ip} + x_{jq} \leq 1 && \text{for all } (i,j) \in A \text{ and } 1 \leq q < p \leq U && \text{(2d)}\\
&\quad z - p\, x_{ip} \geq 0 && \text{for all } i \in V \text{ and } p \in \{1,\dots,U\} && \text{(2e)}\\
&\quad x_{ip} \in \{0,1\} && \text{for all } i \in V \text{ and } p \in \{1,\dots,U\} && \text{(2f)}\\
&\quad z \in \mathbb{Z}_{\geq 0}. && && \text{(2g)}
\end{aligned}
$$

Constraints (2b) ensure that there is exactly one value $p \in \{1,\dots,U\}$ with $x_{ip} = 1$. Constraints (2c) specify that a setup must occur between two operations in distinct classes. Constraints (2d) ensure that all precedence constraints are met. Constraints (2f) state that variables x are binary, and constraint (2g) specifies that the number of setups is integer. Clearly, this last constraint could be dropped. Together with constraints (2e), the objective (2a) expresses the minimization of the total number of setups.

We use Gurobi 7.5 as an IP solver and set the time limit to 7200 s per run. We execute one run for each instance and IP formulation and record the final lower bound, upper bound, and run time. As first observation, we mention that both formulations are not able to adequately tackle the large instances: optimal solutions are found only for few simple instances, which are solved within a few minutes by the branch-and-bound approach, and no feasible solution is obtained for most of the other instances. Hence, the branch-and-bound approach is substantially better for the large instances, and we do not report the detailed results of the IP approaches for the large instances as they have little value. Detailed results for the small instances are given in Table 1 of the appendix.

An analysis of the results for the small instances indicates that IP1 and IP2 solve 146 and 130 instances, respectively, (out of 288) to optimality, and find no feasible solution for 86 and 112 instances, respectively. The high number of instances for which no feasible solution could be found can partially be explained by the tight choice of the upper bound UB: both integer linear programming formulations discard all solutions with objective value larger than U from the solution space. This typically helps to reduce the overall run time, but may increase the difficulty to find a (first) feasible solution.

While the above numbers suggest that IP1 is slightly better than IP2, none of the two is consistently better than the other: 23 and 7 instances are solved to optimality only by IP1 and IP2, respectively. Also, the upper bounds of IP1 and IP2 are quite similar in the instances for which both find a feasible solution.

A deeper analysis indicates that both integer linear programming approaches solve only simple instances. Indeed, each instance solved to optimality by IP1 or IP2 is also solved to optimality by the branch-and-bound algorithm within typically less than 10 seconds. IP1 or IP2 are unable to provide good results for the more difficult instances. This is illustrated in Figure 9, which provides box-and-whisker plots of the RPDs of IP1 and IP2 upper bounds to the upper bound of the branch-and-bound approach for the small dense instances. The portion of instances with no feasible solution is given in brackets. For example, IP1 finds no feasible solution in 22 out of 36 dense instances with 5 classes. For IP2, no box plot is shown for the seven class instances as IP2 does not find any feasible solution for these instances.

We note that no RPD value is negative, which means that the integer linear programming approaches are not able to find better results than the branch-and-bound algorithm. This is no surprise as the branch-and-bound approach solves almost all small instances to optimality. We also observe that both IP1 and IP2 solve all instances with three classes, but are unable to find good solutions in most instances with more than five classes. This can be seen by the large number of instances for which no feasible solution is found (see numbers in brackets) and by the large RPD values.

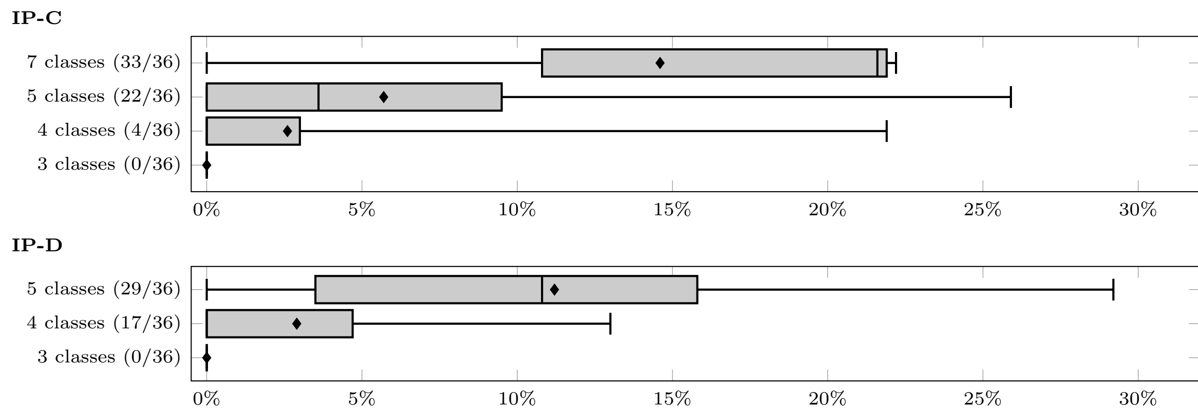


Figure 9: Box plots of the RPDs of IP1 and IP2 upper bounds to the upper bound of the branch-and-bound approach for the small dense instances, grouped by the number of classes.

Altogether, we conclude that the branch-and-bound approach clearly dominates both integer linear programming approaches, which supports our view that the proposed algorithm substantially improves the ability to solve the PCCS problem.

4 Concluding remarks

The PCCS problem can be used to model various recurring decision problems in manufacturing and transportation systems. In scheduling terms, it models a one-machine problem with precedence constraints and setups where the goal is to minimize the number of setups. Previous research has shown that this problem is difficult to solve from both a theoretical and computational perspective. There has been, however, only little research on computational methods.

This article addresses this research gap by proposing a breadth-first-search branch-and-bound approach for the PCCS problem. While the generic branch-and-bound framework is quite standard, the developed approach incorporates sophisticated sub-procedures that successfully exploit the structure of the problem. As a result, the approach gives excellent computational results. It solves many printed circuit board routing instances with up to 350 operations to optimality and provides high-quality solutions and good lower bounds for the other instances. Furthermore, the computational results shed new light on what makes a PCCS instance difficult.

In our view, these developments pave the way for interesting future research. The proposed branch-and-bound approach could be made more efficient by improving its sub-procedures. A more complex heuristic, such as a local search approach similar to the one presented in Meuwly et al. (2010) for mixed graph coloring, could provide a better initial upper bound before starting the actual branch-and-bound search. A promising avenue is also the development of better lower bound calculations. Similar as the one-class-based lower bound, one may calculate a lower bound for each pair of classes. The overall lower bound can then be obtained by choosing and combining some of these lower bounds. When considering the merging procedure, one may try to identify more complex structures to merge such as chains of vertices. From an application perspective, it may be interesting to develop a similar approach to solve exactly the shortest common supersequence problem, which is often used to model applications in genetics.

Appendix

Detailed computational results

Detailed computational results obtained for the small and large instances are given in Table 1 and 2, respectively. The tables are structured as follows. Each row presents the results for one instance. The first five columns describe the instance by providing its name and parameter values W (number of workstations), P (probability for being a small component), D (1 for dense instances, and 0 for non-dense ones), and L (number of large components). The next three columns give the results obtained with our branch-and-bound (B&B) algorithm as follows. Columns LB, UB, and time contain the lower bound, upper bound, and run time (in seconds), respectively. If an optimal solution is found, the values of LB and UB are the same. We therefore omit it in column LB and highlight it in column UB in boldface. For each unsolved instance, the run time is exactly the time limit of 7200 s, and we therefore omit it in column time. In the same way, we report the results obtained with the two integer linear programs in the columns with heading IP1 and IP2. These results are only provided for the small instances.

Table 1: Detailed results for the small instances.

					B&B			IP1			IP2		
Name	**W**	**P**	**D**	**L**	**LB**	**UB**	**time**	**LB**	**UB**	**time**	**LB**	**UB**	**time**
S001	3	0.5	0	0		**26**	1		**26**	19		**26**	43
S002	3	0.5	0	0		**20**	1		**20**	32		**20**	39
S003	3	0.5	0	0		**22**	1		**22**	1204		**22**	198
S004	3	0.5	0	4		**20**	1		**20**	4		**20**	44
S005	3	0.5	0	4		**26**	1		**26**	210		**26**	26
S006	3	0.5	0	4		**16**	1		**16**	14		**16**	45
S007	3	0.5	0	8		**22**	1		**22**	2		**22**	20
S008	3	0.5	0	8		**23**	1		**23**	33		**23**	107
S009	3	0.5	0	8		**16**	1		**16**	27		**16**	33
S010	3	0.5	0	12		**13**	1		**13**	19		**13**	9
S011	3	0.5	0	12		**16**	1		**16**	2		**16**	14
S012	3	0.5	0	12		**13**	1		**13**	14		**13**	8
S013	3	0.5	1	0		**29**	4		**29**	345		**29**	842
S014	3	0.5	1	0		**31**	2		**31**	192		**31**	405
S015	3	0.5	1	0		**28**	2		**28**	468		**28**	891
S016	3	0.5	1	4		**31**	1		**31**	128		**31**	323
S017	3	0.5	1	4		**24**	1		**24**	123		**24**	229
S018	3	0.5	1	4		**26**	1		**26**	261		**26**	583
S019	3	0.5	1	8		**27**	1		**27**	24		**27**	31
S020	3	0.5	1	8		**23**	1		**23**	349		**23**	1120
S021	3	0.5	1	8		**26**	1		**26**	552		**26**	216
S022	3	0.5	1	12		**20**	1		**20**	37		**20**	58
S023	3	0.5	1	12		**20**	1		**20**	6		**20**	15
S024	3	0.5	1	12		**20**	1		**20**	174		**20**	67
S025	3	0.7	0	0		**13**	1		**13**	2		**13**	10
S026	3	0.7	0	0		**9**	1		**9**	95		**9**	6
S027	3	0.7	0	0		**10**	1		**10**	35		**10**	7
S028	3	0.7	0	4		**17**	1		**17**	15		**17**	12
S029	3	0.7	0	4		**9**	1		**9**	18		**9**	6
S030	3	0.7	0	4		**9**	1		**9**	23		**9**	7
S031	3	0.7	0	8		**12**	1		**12**	45		**12**	54
S032	3	0.7	0	8		**9**	1		**9**	95		**9**	10
S033	3	0.7	0	8		**16**	1		**16**	9		**16**	12
S034	3	0.7	0	12		**11**	1		**11**	252		**11**	17
S035	3	0.7	0	12		**10**	1		**10**	2		**10**	4
S036	3	0.7	0	12		**9**	1		**9**	9		**9**	14
S037	3	0.7	1	0		**25**	4		**25**	4496		**25**	2026
S038	3	0.7	1	0		**26**	4		**26**	766		**26**	3525
S039	3	0.7	1	0		**27**	3		**27**	292		**27**	576
S040	3	0.7	1	4		**26**	2		**26**	349		**26**	2663
S041	3	0.7	1	4		**25**	1		**25**	532		**25**	2057
S042	3	0.7	1	4		**23**	1		**23**	135		**23**	1140
S043	3	0.7	1	8		**22**	1		**22**	1007		**22**	279
S044	3	0.7	1	8		**24**	1		**24**	184		**24**	452

Table 1: continued

Name	W	P	D	L	B&B LB	B&B UB	B&B time	IP1 LB	IP1 UB	IP1 time	IP2 LB	IP2 UB	IP2 time
S045	3	0.7	1	8		**24**	1		**24**	239		**24**	393
S046	3	0.7	1	12		**13**	1		**13**	77		**13**	40
S047	3	0.7	1	12		**17**	1		**17**	34		**17**	72
S048	3	0.7	1	12		**19**	1		**19**	330		**19**	587
S049	3	0.9	0	0		**6**	1		**6**	5		**6**	4
S050	3	0.9	0	0		**7**	1		**7**	12		**7**	6
S051	3	0.9	0	0		**6**	1		**6**	2		**6**	3
S052	3	0.9	0	4		**6**	1		**6**	13		**6**	28
S053	3	0.9	0	4		**7**	1		**7**	7		**7**	3
S054	3	0.9	0	4		**6**	1		**6**	8		**6**	3
S055	3	0.9	0	8		**6**	1		**6**	7		**6**	4
S056	3	0.9	0	8		**6**	1		**6**	12		**6**	10
S057	3	0.9	0	8		**6**	1		**6**	6		**6**	6
S058	3	0.9	0	12		**6**	1		**6**	11		**6**	16
S059	3	0.9	0	12		**6**	1		**6**	7		**6**	2
S060	3	0.9	0	12		**6**	1		**6**	7		**6**	3
S061	3	0.9	1	0		**24**	5		**24**	4024		**24**	2045
S062	3	0.9	1	0		**22**	2		**22**	1526		**22**	2191
S063	3	0.9	1	0		**23**	4		**23**	5138		**23**	2319
S064	3	0.9	1	4		**16**	1		**16**	38		**16**	102
S065	3	0.9	1	4		**21**	2		**21**	396		**21**	1314
S066	3	0.9	1	4		**22**	2		**22**	551		**22**	2680
S067	3	0.9	1	8		**16**	1		**16**	659		**16**	352
S068	3	0.9	1	8		**18**	1		**18**	462		**18**	222
S069	3	0.9	1	8		**17**	1		**17**	18		**17**	51
S070	3	0.9	1	12		**15**	1		**15**	2		**15**	8
S071	3	0.9	1	12		**12**	1		**12**	16		**12**	6
S072	3	0.9	1	12		**18**	1		**18**	336		**18**	848
S073	4	0.5	0	0		**30**	2		**30**	3156	1	-	
S074	4	0.5	0	0		**33**	1		**33**	1409		**33**	3040
S075	4	0.5	0	0		**26**	2		**26**	3224		**26**	5393
S076	4	0.5	0	4		**26**	1		**26**	451		**26**	3763
S077	4	0.5	0	4		**26**	1		**26**	1113		**26**	1349
S078	4	0.5	0	4		**26**	1		**26**	5721		**26**	5552
S079	4	0.5	0	8		**19**	1		**19**	756		**19**	6590
S080	4	0.5	0	8		**20**	1		**20**	751		**20**	3686
S081	4	0.5	0	8		**18**	1		**18**	2636		**18**	2413
S082	4	0.5	0	12		**20**	1		**20**	1511		**20**	2848
S083	4	0.5	0	12		**19**	1		**19**	519		**19**	3210
S084	4	0.5	0	12		**16**	1		**16**	668		**16**	2480
S085	4	0.5	1	0		**41**	38	36	42		17	-	
S086	4	0.5	1	0		**37**	28	35	37		33	-	
S087	4	0.5	1	0		**35**	88	30	36		16	-	
S088	4	0.5	1	4		**32**	8	31	32		24	-	
S089	4	0.5	1	4		**36**	40	32	38		32	-	
S090	4	0.5	1	4		**38**	23	34	38		33	-	
S091	4	0.5	1	8		**31**	2		**31**	3096		**31**	6765
S092	4	0.5	1	8		**27**	2		**27**	880	24	-	
S093	4	0.5	1	8		**24**	5	20	26		20	25	
S094	4	0.5	1	12		**27**	1	24	27			**27**	4158
S095	4	0.5	1	12		**21**	2		**21**	1290		**21**	1470
S096	4	0.5	1	12		**31**	3		**31**	3116		**31**	6156
S097	4	0.7	0	0		**20**	1		**20**	686		**20**	1659
S098	4	0.7	0	0		**10**	1		**10**	709		**10**	2083
S099	4	0.7	0	0		**21**	1	18	21			**21**	4408
S100	4	0.7	0	4		**19**	1		**19**	366		**19**	1623
S101	4	0.7	0	4		**10**	1		**10**	1205		**10**	932
S102	4	0.7	0	4		**17**	1		**17**	347		**17**	221
S103	4	0.7	0	8		**13**	1		**13**	803		**13**	295
S104	4	0.7	0	8		**13**	1		**13**	609		**13**	913
S105	4	0.7	0	8		**12**	1		**12**	605		**12**	1723
S106	4	0.7	0	12		**15**	1		**15**	314		**15**	3070
S107	4	0.7	0	12		**13**	1		**13**	257		**13**	1079
S108	4	0.7	0	12		**13**	1		**13**	263		**13**	1235
S109	4	0.7	1	0		**32**	88	24	-		11	-	

Table 1: continued

					B&B			IP1			IP2		
Name	**W**	**P**	**D**	**L**	**LB**	**UB**	**time**	**LB**	**UB**	**time**	**LB**	**UB**	**time**
S110	4	0.7	1	0		**32**	79	26	-		26	-	
S111	4	0.7	1	0		**36**	137	32	37		28	-	
S112	4	0.7	1	4		**29**	17	25	31		28	30	
S113	4	0.7	1	4		**29**	20	25	30		25	-	
S114	4	0.7	1	4		**31**	22	26	31		27	-	
S115	4	0.7	1	8		**25**	4	21	25			**25**	4722
S116	4	0.7	1	8		**22**	3		**22**	4681		**22**	3867
S117	4	0.7	1	8		**25**	5	21	28		23	26	
S118	4	0.7	1	12		**19**	2		**19**	4438	18	20	
S119	4	0.7	1	12		**26**	5		**26**	2316	25	26	
S120	4	0.7	1	12		**23**	3	19	23		20	23	
S121	4	0.9	0	0		**7**	1		**7**	42		**7**	142
S122	4	0.9	0	0		**7**	1		**7**	731		**7**	1518
S123	4	0.9	0	0		**8**	1		**8**	74		**8**	215
S124	4	0.9	0	4		**8**	1		**8**	120		**8**	161
S125	4	0.9	0	4		**9**	1		**9**	634		**9**	406
S126	4	0.9	0	4		**9**	1		**9**	995		**9**	474
S127	4	0.9	0	8		**9**	1		**9**	542		**9**	370
S128	4	0.9	0	8		**9**	1		**9**	449		**9**	1807
S129	4	0.9	0	8		**8**	1		**8**	1493		**8**	2574
S130	4	0.9	0	12		**8**	1		**8**	1833		**8**	437
S131	4	0.9	0	12		**9**	1		**9**	398		**9**	67
S132	4	0.9	0	12		**10**	1		**10**	444		**10**	2432
S133	4	0.9	1	0		**30**	72	22	-		12	-	
S134	4	0.9	1	0		**30**	86	24	-		22	-	
S135	4	0.9	1	0		**32**	135	24	39		8	-	
S136	4	0.9	1	4		**23**	10		**23**	3599	20	26	
S137	4	0.9	1	4		**29**	30	21	32		22	-	
S138	4	0.9	1	4		**27**	14	24	27		21	-	
S139	4	0.9	1	8		**24**	3		**24**	4288	19	26	
S140	4	0.9	1	8		**21**	3		**21**	3740	18	23	
S141	4	0.9	1	8		**25**	3	20	27		21	27	
S142	4	0.9	1	12		**22**	2		**22**	810		**22**	3407
S143	4	0.9	1	12		**23**	3	20	23			**23**	3137
S144	4	0.9	1	12		**24**	2		**24**	391		**24**	3596
S145	5	0.5	0	0		**25**	3	22	27		22	27	
S146	5	0.5	0	0		**29**	2	24	31		25	-	
S147	5	0.5	0	0		**34**	6	31	34		14	-	
S148	5	0.5	0	4		**24**	3	19	25		17	-	
S149	5	0.5	0	4		**36**	12		**36**	1602	28	42	
S150	5	0.5	0	4		**22**	5		**22**	3479	19	25	
S151	5	0.5	0	8		**30**	2	28	31			**30**	5814
S152	5	0.5	0	8		**25**	1		**25**	1839		**25**	6676
S153	5	0.5	0	8		**26**	4		**26**	1476	24	26	
S154	5	0.5	0	12		**24**	1		**24**	1303		**24**	4915
S155	5	0.5	0	12		**24**	2	20	25		20	26	
S156	5	0.5	0	12		**24**	2		**24**	343		**24**	1080
S157	5	0.5	1	0		**46**	137	39	-		37	-	
S158	5	0.5	1	0		**45**	212	37	-		19	-	
S159	5	0.5	1	0		**45**	1265	33	-		31	-	
S160	5	0.5	1	4		**39**	209	31	-		9	-	
S161	5	0.5	1	4		**40**	115	34	-		14	-	
S162	5	0.5	1	4		**42**	134	33	-		31	-	
S163	5	0.5	1	8		**36**	32	28	-		12	-	
S164	5	0.5	1	8		**36**	64	28	-		29	-	
S165	5	0.5	1	8		**35**	122	27	39		25	-	
S166	5	0.5	1	12		**32**	34	28	32		10	-	
S167	5	0.5	1	12		**32**	16	30	32		29	33	
S168	5	0.5	1	12		**37**	48	32	40		31	41	
S169	5	0.7	0	0		**15**	1	10	17			**15**	6656
S170	5	0.7	0	0		**15**	1		**15**	2958	13	15	
S171	5	0.7	0	0		**15**	1		**15**	6941	12	-	
S172	5	0.7	0	4		**20**	2		**20**	5394	16	22	
S173	5	0.7	0	4		**15**	1		**15**	3367	12	-	
S174	5	0.7	0	4		**21**	1		**21**	6238	18	24	

Table 1: continued

					B&B			IP1			IP2		
Name	**W**	**P**	**D**	**L**	**LB**	**UB**	**time**	**LB**	**UB**	**time**	**LB**	**UB**	**time**
S175	5	0.7	0	8		**18**	1		**18**	6471	16	18	
S176	5	0.7	0	8		**13**	1		**13**	1316		**13**	5017
S177	5	0.7	0	8		**16**	1		**16**	3534	13	19	
S178	5	0.7	0	12		**14**	1		**14**	2034		**14**	6213
S179	5	0.7	0	12		**13**	1	10	14			**13**	6524
S180	5	0.7	0	12		**13**	1	10	17		11	14	
S181	5	0.7	1	0		**39**	1193	25	-		7	-	
S182	5	0.7	1	0		**38**	1691	23	-		6	-	
S183	5	0.7	1	0		**39**	1544	25	-		9	-	
S184	5	0.7	1	4		**35**	424	26	-		19	-	
S185	5	0.7	1	4		**35**	205	26	-		20	-	
S186	5	0.7	1	4		**37**	644	27	-		11	-	
S187	5	0.7	1	8		**26**	43	23	27		20	27	
S188	5	0.7	1	8		**29**	105		**29**	4722	22	-	
S189	5	0.7	1	8		**29**	13	21	30		21	-	
S190	5	0.7	1	12		**28**	14	24	28		24	28	
S191	5	0.7	1	12		**30**	38		**30**	4990	23	35	
S192	5	0.7	1	12		**24**	6		**24**	4034	21	31	
S193	5	0.9	0	0		**10**	1	7	-		8	10	
S194	5	0.9	0	0		**12**	1		**12**	1125	11	12	
S195	5	0.9	0	0		**8**	1		**8**	3744		**8**	1010
S196	5	0.9	0	4		**10**	1		**10**	1987	7	10	
S197	5	0.9	0	4		**9**	1		**9**	1932	8	10	
S198	5	0.9	0	4		**8**	1		**8**	2749		**8**	2594
S199	5	0.9	0	8		**9**	1	8	10		7	10	
S200	5	0.9	0	8		**9**	1		**9**	1450		**9**	3695
S201	5	0.9	0	8		**9**	1	7	10		8	10	
S202	5	0.9	0	12		**10**	1		**10**	1926		**10**	6085
S203	5	0.9	0	12		**9**	1		**9**	1237		**9**	1356
S204	5	0.9	0	12		**11**	1		**11**	1057		**11**	709
S205	5	0.9	1	0		**37**	928	22	-		11	-	
S206	5	0.9	1	0		**35**	723	22	-		7	-	
S207	5	0.9	1	0		**34**	1806	22	-		7	-	
S208	5	0.9	1	4		**30**	199	21	-		21	-	
S209	5	0.9	1	4		**33**	404	25	-		8	-	
S210	5	0.9	1	4		**32**	163	24	-		9	-	
S211	5	0.9	1	8		**26**	24	19	-		19	-	
S212	5	0.9	1	8		**30**	140	22	-		21	-	
S213	5	0.9	1	8		**27**	71	22	34		22	-	
S214	5	0.9	1	12		**24**	9	19	27		18	-	
S215	5	0.9	1	12		**23**	11	19	24		19	-	
S216	5	0.9	1	12		**20**	11	16	22		15	23	
S217	7	0.5	0	0		**31**	184	19	-		17	-	
S218	7	0.5	0	0		**43**	123	31	-		30	-	
S219	7	0.5	0	0		**39**	118	30	-		21	-	
S220	7	0.5	0	4		**31**	134	21	-		21	-	
S221	7	0.5	0	4		**27**	30	16	-		18	-	
S222	7	0.5	0	4		**36**	57	29	-		9	-	
S223	7	0.5	0	8		**26**	20	19	-		18	-	
S224	7	0.5	0	8		**30**	21	23	-		22	-	
S225	7	0.5	0	8		**31**	171	21	-		22	-	
S226	7	0.5	0	12		**24**	32	16	28		16	-	
S227	7	0.5	0	12		**26**	94	19	-		17	-	
S228	7	0.5	0	12		**25**	10	18	-		16	-	
S229	7	0.5	1	0	49	57		37	-		18	-	
S230	7	0.5	1	0		**51**	7036	30	-		23	-	
S231	7	0.5	1	0		**54**	1782	37	-		19	-	
S232	7	0.5	1	4	40	48		33	-		24	-	
S233	7	0.5	1	4		**51**	868	33	-		21	-	
S234	7	0.5	1	4		**53**	895	36	-		28	-	
S235	7	0.5	1	8		**45**	2452	32	-		28	-	
S236	7	0.5	1	8		**38**	457	28	-		9	-	
S237	7	0.5	1	8		**45**	774	30	-		29	-	
S238	7	0.5	1	12		**36**	171	28	-		27	-	
S239	7	0.5	1	12		**45**	576	34	-		32	-	

Table 1: continued

Name	W	P	D	L	B&B			IP1			IP2		
					LB	UB	time	LB	UB	time	LB	UB	time
S240	7	0.5	1	12		**44**	328	34	-		26	-	
S241	7	0.7	0	0		**23**	2	15	-		15	-	
S242	7	0.7	0	0		**25**	20	14	-		15	-	
S243	7	0.7	0	0		**17**	1	8	-		10	-	
S244	7	0.7	0	4		**25**	17	14	-		13	-	
S245	7	0.7	0	4		**20**	3	11	-		11	-	
S246	7	0.7	0	4		**21**	3	13	-		12	25	
S247	7	0.7	0	8		**21**	5	13	-		15	-	
S248	7	0.7	0	8		**21**	2	15	-		15	-	
S249	7	0.7	0	8		**15**	1	8	-		11	-	
S250	7	0.7	0	12		**18**	3	11	-		13	-	
S251	7	0.7	0	12		**20**	2	12	-		14	-	
S252	7	0.7	0	12		**17**	1	10	21		12	-	
S253	7	0.7	1	0	37	50		25	-		6	-	
S254	7	0.7	1	0	40	53		31	-		5	-	
S255	7	0.7	1	0	39	51		25	-		21	-	
S256	7	0.7	1	4	36	47		27	-		18	-	
S257	7	0.7	1	4	34	44		24	-		20	-	
S258	7	0.7	1	4		**41**	5555	24	-		20	-	
S259	7	0.7	1	8	31	38		23	-		20	-	
S260	7	0.7	1	8		**34**	524	25	-		24	-	
S261	7	0.7	1	8		**36**	1004	27	44		24	-	
S262	7	0.7	1	12		**30**	169	24	30		23	-	
S263	7	0.7	1	12		**30**	641	20	-		18	-	
S264	7	0.7	1	12		**37**	2111	24	45		24	-	
S265	7	0.9	0	0		**13**	1	7	15		9	14	
S266	7	0.9	0	0		**13**	1	7	-		8	13	
S267	7	0.9	0	0		**14**	1	8	-		8	15	
S268	7	0.9	0	4		**15**	1	8	17		9	16	
S269	7	0.9	0	4		**14**	1	8	16		10	16	
S270	7	0.9	0	4		**12**	1	7	-		9	-	
S271	7	0.9	0	8		**12**	1	6	13		9	12	
S272	7	0.9	0	8		**13**	1	7	15		10	15	
S273	7	0.9	0	8		**13**	1	9	-		11	13	
S274	7	0.9	0	12		**13**	1	7	14		10	14	
S275	7	0.9	0	12		**11**	1	7	13		10	11	
S276	7	0.9	0	12		**13**	1	7	14		9	13	
S277	7	0.9	1	0	38	50		23	-		4	-	
S278	7	0.9	1	0	37	50		22	-		18	-	
S279	7	0.9	1	0	37	47		25	-		6	-	
S280	7	0.9	1	4	31	40		22	-		11	-	
S281	7	0.9	1	4		**33**	4268	22	-		21	-	
S282	7	0.9	1	4	33	45		23	-		13	-	
S283	7	0.9	1	8		**33**	5470	18	-		19	-	
S284	7	0.9	1	8		**40**	6055	23	-		19	-	
S285	7	0.9	1	8		**36**	4232	23	-		8	-	
S286	7	0.9	1	12		**29**	646	15	-		17	-	
S287	7	0.9	1	12		**31**	109	22	-		21	-	
S288	7	0.9	1	12		**29**	618	18	-		19	-	

Table 2: Detailed results for the large instances.

Name	W	P	D	L	B&B		
					LB	UB	time
L001	3	0.5	0	0		**26**	3
L002	3	0.5	0	0		**36**	3
L003	3	0.5	0	0		**32**	8
L004	3	0.5	0	4		**27**	3
L005	3	0.5	0	4		**24**	3
L006	3	0.5	0	4		**30**	2
L007	3	0.5	0	8		**23**	1
L008	3	0.5	0	8		**21**	1

Name	W	P	D	L	B&B		
					LB	UB	time
L145	5	0.5	0	0	42	50	
L146	5	0.5	0	0		**34**	2023
L147	5	0.5	0	0		**43**	3247
L148	5	0.5	0	4		**40**	4571
L149	5	0.5	0	4		**47**	866
L150	5	0.5	0	4		**43**	532
L151	5	0.5	0	8		**41**	5972
L152	5	0.5	0	8		**49**	614

Table 2: continued

Name	W	P	D	L	B&B LB	B&B UB	B&B time
L009	3	0.5	0	8		**23**	2
L010	3	0.5	0	12		**29**	1
L011	3	0.5	0	12		**22**	2
L012	3	0.5	0	12		**29**	2
L013	3	0.5	1	0		**45**	202
L014	3	0.5	1	0		**47**	364
L015	3	0.5	1	0		**45**	182
L016	3	0.5	1	4		**47**	457
L017	3	0.5	1	4		**47**	171
L018	3	0.5	1	4		**39**	117
L019	3	0.5	1	8		**43**	115
L020	3	0.5	1	8		**48**	112
L021	3	0.5	1	8		**46**	45
L022	3	0.5	1	12		**45**	68
L023	3	0.5	1	12		**40**	71
L024	3	0.5	1	12		**40**	84
L025	3	0.7	0	0		**27**	1
L026	3	0.7	0	0		**12**	1
L027	3	0.7	0	0		**14**	1
L028	3	0.7	0	4		**15**	1
L029	3	0.7	0	4		**12**	1
L030	3	0.7	0	4		**14**	1
L031	3	0.7	0	8		**12**	1
L032	3	0.7	0	8		**13**	1
L033	3	0.7	0	8		**12**	1
L034	3	0.7	0	12		**15**	1
L035	3	0.7	0	12		**14**	1
L036	3	0.7	0	12		**11**	1
L037	3	0.7	1	0		**38**	295
L038	3	0.7	1	0		**38**	285
L039	3	0.7	1	0		**42**	301
L040	3	0.7	1	4		**35**	158
L041	3	0.7	1	4		**34**	262
L042	3	0.7	1	4		**34**	118
L043	3	0.7	1	8		**34**	90
L044	3	0.7	1	8		**39**	264
L045	3	0.7	1	8		**36**	238
L046	3	0.7	1	12		**33**	21
L047	3	0.7	1	12		**29**	40
L048	3	0.7	1	12		**35**	16
L049	3	0.9	0	0		**8**	1
L050	3	0.9	0	0		**7**	1
L051	3	0.9	0	0		**7**	1
L052	3	0.9	0	4		**8**	1
L053	3	0.9	0	4		**6**	1
L054	3	0.9	0	4		**7**	1
L055	3	0.9	0	8		**9**	1
L056	3	0.9	0	8		**7**	1
L057	3	0.9	0	8		**8**	1
L058	3	0.9	0	12		**7**	1
L059	3	0.9	0	12		**8**	1
L060	3	0.9	0	12		**8**	1
L061	3	0.9	1	0		**34**	780
L062	3	0.9	1	0		**34**	689
L063	3	0.9	1	0		**36**	615
L064	3	0.9	1	4		**33**	222
L065	3	0.9	1	4		**32**	53
L066	3	0.9	1	4		**33**	89
L067	3	0.9	1	8		**30**	76
L068	3	0.9	1	8		**29**	12
L069	3	0.9	1	8		**32**	30
L070	3	0.9	1	12		**30**	7
L071	3	0.9	1	12		**28**	19
L072	3	0.9	1	12		**27**	20
L073	4	0.5	0	0		**45**	20
L153	5	0.5	0	8		**35**	354
L154	5	0.5	0	12		**40**	2081
L155	5	0.5	0	12		**50**	1022
L156	5	0.5	0	12		**34**	3385
L157	5	0.5	1	0	64	85	
L158	5	0.5	1	0	58	81	
L159	5	0.5	1	0	64	79	
L160	5	0.5	1	4	60	79	
L161	5	0.5	1	4	54	74	
L162	5	0.5	1	4	54	71	
L163	5	0.5	1	8	50	67	
L164	5	0.5	1	8	52	69	
L165	5	0.5	1	8	54	65	
L166	5	0.5	1	12	45	60	
L167	5	0.5	1	12	48	65	
L168	5	0.5	1	12	53	70	
L169	5	0.7	0	0		**22**	9
L170	5	0.7	0	0		**24**	40
L171	5	0.7	0	0		**22**	40
L172	5	0.7	0	4		**18**	4
L173	5	0.7	0	4		**22**	10
L174	5	0.7	0	4		**19**	4
L175	5	0.7	0	8		**21**	11
L176	5	0.7	0	8		**20**	3
L177	5	0.7	0	8		**21**	5
L178	5	0.7	0	12		**21**	8
L179	5	0.7	0	12		**21**	10
L180	5	0.7	0	12		**19**	11
L181	5	0.7	1	0	48	66	
L182	5	0.7	1	0	45	64	
L183	5	0.7	1	0	51	72	
L184	5	0.7	1	4	45	68	
L185	5	0.7	1	4	44	61	
L186	5	0.7	1	4	42	61	
L187	5	0.7	1	8	39	55	
L188	5	0.7	1	8	41	57	
L189	5	0.7	1	8	39	54	
L190	5	0.7	1	12	38	51	
L191	5	0.7	1	12	42	58	
L192	5	0.7	1	12	38	52	
L193	5	0.9	0	0		**12**	1
L194	5	0.9	0	0		**13**	1
L195	5	0.9	0	0		**11**	1
L196	5	0.9	0	4		**12**	1
L197	5	0.9	0	4		**10**	1
L198	5	0.9	0	4		**12**	1
L199	5	0.9	0	8		**11**	1
L200	5	0.9	0	8		**10**	1
L201	5	0.9	0	8		**11**	1
L202	5	0.9	0	12		**13**	1
L203	5	0.9	0	12		**13**	1
L204	5	0.9	0	12		**11**	1
L205	5	0.9	1	0	40	61	
L206	5	0.9	1	0	40	61	
L207	5	0.9	1	0	40	61	
L208	5	0.9	1	4	41	57	
L209	5	0.9	1	4	35	53	
L210	5	0.9	1	4	36	54	
L211	5	0.9	1	8	36	56	
L212	5	0.9	1	8	41	55	
L213	5	0.9	1	8	35	48	
L214	5	0.9	1	12	35	49	
L215	5	0.9	1	12	33	44	
L216	5	0.9	1	12	37	48	
L217	7	0.5	0	0	38	49	

Table 2: continued

Name	W	P	D	L	B&B LB	B&B UB	B&B time
L074	4	0.5	0	0		**33**	89
L075	4	0.5	0	0		**32**	218
L076	4	0.5	0	4		**36**	86
L077	4	0.5	0	4		**30**	149
L078	4	0.5	0	4		**31**	97
L079	4	0.5	0	8		**37**	104
L080	4	0.5	0	8		**27**	40
L081	4	0.5	0	8		**41**	172
L082	4	0.5	0	12		**35**	171
L083	4	0.5	0	12		**29**	23
L084	4	0.5	0	12		**37**	18
L085	4	0.5	1	0	50	63	
L086	4	0.5	1	0	51	62	
L087	4	0.5	1	0	54	62	
L088	4	0.5	1	4	48	58	
L089	4	0.5	1	4	51	59	
L090	4	0.5	1	4	48	56	
L091	4	0.5	1	8	47	55	
L092	4	0.5	1	8		**51**	3054
L093	4	0.5	1	8	47	57	
L094	4	0.5	1	12	43	53	
L095	4	0.5	1	12	43	54	
L096	4	0.5	1	12		**50**	3897
L097	4	0.7	0	0		**18**	2
L098	4	0.7	0	0		**24**	2
L099	4	0.7	0	0		**19**	1
L100	4	0.7	0	4		**19**	1
L101	4	0.7	0	4		**20**	1
L102	4	0.7	0	4		**15**	1
L103	4	0.7	0	8		**14**	1
L104	4	0.7	0	8		**19**	2
L105	4	0.7	0	8		**18**	2
L106	4	0.7	0	12		**19**	1
L107	4	0.7	0	12		**24**	3
L108	4	0.7	0	12		**15**	1
L109	4	0.7	1	0	38	49	
L110	4	0.7	1	0	43	53	
L111	4	0.7	1	0	39	52	
L112	4	0.7	1	4	42	50	
L113	4	0.7	1	4	41	51	
L114	4	0.7	1	4	38	49	
L115	4	0.7	1	8	35	44	
L116	4	0.7	1	8	39	48	
L117	4	0.7	1	8		**40**	6023
L118	4	0.7	1	12		**41**	2920
L119	4	0.7	1	12		**43**	5945
L120	4	0.7	1	12	38	44	
L121	4	0.9	0	0		**8**	1
L122	4	0.9	0	0		**9**	1
L123	4	0.9	0	0		**9**	1
L124	4	0.9	0	4		**12**	1
L125	4	0.9	0	4		**11**	1
L126	4	0.9	0	4		**11**	1
L127	4	0.9	0	8		**11**	1
L128	4	0.9	0	8		**10**	1
L129	4	0.9	0	8		**9**	1
L130	4	0.9	0	12		**10**	1
L131	4	0.9	0	12		**9**	1
L132	4	0.9	0	12		**11**	1
L133	4	0.9	1	0	37	49	
L134	4	0.9	1	0	35	45	
L135	4	0.9	1	0	36	46	
L136	4	0.9	1	4	36	45	
L137	4	0.9	1	4	35	43	
L138	4	0.9	1	4	36	46	

Name	W	P	D	L	B&B LB	B&B UB	B&B time
L218	7	0.5	0	0	45	63	
L219	7	0.5	0	0	49	65	
L220	7	0.5	0	4	61	79	
L221	7	0.5	0	4	59	75	
L222	7	0.5	0	4	34	44	
L223	7	0.5	0	8	40	54	
L224	7	0.5	0	8	38	52	
L225	7	0.5	0	8	45	60	
L226	7	0.5	0	12	42	57	
L227	7	0.5	0	12	47	61	
L228	7	0.5	0	12	44	56	
L229	7	0.5	1	0	66	103	
L230	7	0.5	1	0	68	105	
L231	7	0.5	1	0	70	103	
L232	7	0.5	1	4	55	89	
L233	7	0.5	1	4	64	101	
L234	7	0.5	1	4	75	103	
L235	7	0.5	1	8	62	98	
L236	7	0.5	1	8	65	93	
L237	7	0.5	1	8	67	96	
L238	7	0.5	1	12	65	92	
L239	7	0.5	1	12	61	89	
L240	7	0.5	1	12	61	88	
L241	7	0.7	0	0		**27**	876
L242	7	0.7	0	0		**27**	803
L243	7	0.7	0	0		**28**	241
L244	7	0.7	0	4		**30**	633
L245	7	0.7	0	4		**24**	134
L246	7	0.7	0	4		**25**	258
L247	7	0.7	0	8		**27**	592
L248	7	0.7	0	8		**25**	86
L249	7	0.7	0	8		**29**	1110
L250	7	0.7	0	12		**25**	193
L251	7	0.7	0	12		**26**	59
L252	7	0.7	0	12		**23**	58
L253	7	0.7	1	0	57	91	
L254	7	0.7	1	0	55	92	
L255	7	0.7	1	0	52	87	
L256	7	0.7	1	4	54	83	
L257	7	0.7	1	4	51	82	
L258	7	0.7	1	4	51	84	
L259	7	0.7	1	8	47	80	
L260	7	0.7	1	8	51	79	
L261	7	0.7	1	8	48	75	
L262	7	0.7	1	12	44	71	
L263	7	0.7	1	12	49	72	
L264	7	0.7	1	12	43	72	
L265	7	0.9	0	0		**15**	2
L266	7	0.9	0	0		**14**	1
L267	7	0.9	0	0		**14**	1
L268	7	0.9	0	4		**14**	1
L269	7	0.9	0	4		**14**	1
L270	7	0.9	0	4		**15**	1
L271	7	0.9	0	8		**14**	1
L272	7	0.9	0	8		**18**	5
L273	7	0.9	0	8		**18**	2
L274	7	0.9	0	12		**16**	2
L275	7	0.9	0	12		**14**	1
L276	7	0.9	0	12		**15**	1
L277	7	0.9	1	0	44	85	
L278	7	0.9	1	0	46	83	
L279	7	0.9	1	0	47	85	
L280	7	0.9	1	4	41	67	
L281	7	0.9	1	4	46	75	
L282	7	0.9	1	4	49	77	

Table 2: continued

Name	W	P	D	L	B&B LB	B&B UB	B&B time	Name	W	P	D	L	B&B LB	B&B UB	B&B time
L139	4	0.9	1	8	32	40		L283	7	0.9	1	8	46	76	
L140	4	0.9	1	8	33	42		L284	7	0.9	1	8	40	65	
L141	4	0.9	1	8		**44**	5947	L285	7	0.9	1	8	40	75	
L142	4	0.9	1	12		**35**	2092	L286	7	0.9	1	12	36	63	
L143	4	0.9	1	12		**37**	3740	L287	7	0.9	1	12	39	57	
L144	4	0.9	1	12	33	38		L288	7	0.9	1	12	36	59	

References

A. V. Aho, M. R. Garey, and J. D. Ullman. The transitive reduction of a directed graph. SIAM Journal on Computing, 1(2):131–137, 1972.

G. V. Andreev, Y. N. Sotskov, and F. Werner. A branch and bound method for mixed graph coloring and scheduling. In Proceedings of the 16th International Conference on CAD/CAM Robotics & Factories of the Future, volume 1, pages 1–8, Trinidad and Tobago, 2000.

I. Blöchliger and N. Zufferey. A graph coloring heuristic using partial solutions and a reactive tabu scheme. Computers and Operations Research, 35(3):960–975, 2008.

A. Camelot. Modélisation des tâches pour la récupération de pièces réutilisables sur un avion en fin de vie. Master's thesis, Department of Mechanical Engineering, Polytechnique Montréal, 2012.

J. R. Correa, S. Fiorini, and N. E. Stier-Moses. A note on the precedence-constrained class sequencing problem. Discrete Applied Mathematics, 155(3):257–259, 2007.

A. Darte. On the complexity of loop fusion. Parallel Computing, 26(9):1175–1193, 2000.

P. Hansen, J. Kuplinsky, and D. Werra. Mixed graph colorings. Mathematical Methods of Operations Research, 45(1): 145–160, 1997.

A. Kouider, H. Ait Haddadène, S. Ourari, and A. Oulamara. Mixed graph colouring for unit-time scheduling. International Journal of Production Research, 55(6):1720–1729, 2017.

C. B. Lofgren. Machine configuration of flexible printed circuit board assembly systems. PhD thesis, School of ISyE, Georgia Institute of Technology, 1986.

C. B. Lofgren, F. M. Leon, and C. A. Tovey. Routing printed circuit cards through an assembly cell. Operations Research, 39(6):992–1004, 1991.

F.-X. Meuwly, B. Ries, and N. Zufferey. Solution methods for a scheduling problem with incompatibility and precedence constraints. Algorithmic Operations Research, 5(2):75–85, 2010.

C. H. Papadimitriou and K. Steiglitz. Combinatorial Optimization: Algorithms and Complexity. Dover Publications, 1998.

B. Ries. Coloring some classes of mixed graphs. Discrete Applied Mathematics, 155(1):1–6, 2007.

Y. N. Sotskov, V. S. Tanaev, and F. Werner. Scheduling problems and mixed graph colorings. Optimization, 51(3): 597–624, 2002.

C. A. Tovey. Non-approximability of precedence-constrained sequencing to minimize setups. Discrete Applied Mathematics, 134(1–3):351–360, 2004.