KU LEUVEN

FACULTY OF ECONOMICS AND BUSINESS

An exact algorithm for parallel machine scheduling with conflicts

Kowalczyk D, Leus R.

KBI_1505

An exact algorithm for parallel machine scheduling with conflicts

Daniel Kowalczyk, Roel Leus
ORSTAT, Faculty of Economics and Business, KU Leuven, Leuven, Belgium
daniel.kowalczyk@kuleuven.be, roel.leus@kuleuven.be

We consider an extension of classic parallel machine scheduling where a set of jobs is scheduled on identical parallel machines and an undirected conflict graph is part of the input. Each node in the graph represents a job and an edge implies that its two jobs are conflicting, meaning that they cannot be scheduled on the same machine. The goal is to find an assignment of the jobs to the machines such that the maximum completion time (makespan) is minimized. We present an exact algorithm based on branch and price that combines methods from bin packing, scheduling and graph coloring, with appropriate modifications. The algorithm has a good computational performance even for parallel machine scheduling without conflicting jobs.

Key words: scheduling; combinatorial optimization; parallel machines; coloring; branch and price

1. Introduction

We schedule a set $J = \{1, \ldots, n\}$ of n independent jobs on m identical parallel machines without preemption such that the maximum completion time of the jobs, or makespan, is minimized. Each job j has an associated processing time $p_j \in \mathbb{N}_0$ and is to be assigned to a single machine. We assume that the processing times are sorted so that $p_1 \geq p_2 \geq \cdots \geq p_n$. The machines are gathered in set $M = \{1, \ldots, m\}$ and each machine can process at most one job at a time. An undirected graph $G = (J, E)$, subsequently referred to as *conflict graph*, is part of the input. If $\{j, j'\} \in E$ then jobs j and j' are *conflicting jobs*, and they need to be assigned to different machines. We call the resulting problem the parallel machine scheduling problem with conflicts (PMC). This problem is NP-hard because it contains both $P||C_{\max}$ (in the standard three-field notation of Lawler et al. 1982) as well as the vertex coloring problem (VCP) as special cases. A feasible schedule exists if and only if the conflict graph can be colored with at most m colors; we will assume $m < n$ to avoid trivial solutions. Moreover, VCP is hard to approximate, and it can turn out to be hard to quickly find even a feasible schedule for a given instance. We conclude that PMC combines two very hard problems.

PMC is theoretically important because it generalizes two well-known problems in combinatorial optimization, but it also naturally arises as (sub-)problem in a number of practical

applications in multiprocessor scheduling, TV advertisement scheduling and audit scheduling. Concretely, PMC is for instance a subproblem of scheduling computing services on different machines: see the ROADEF/EURO Challenge 2012 (ROADEF 2011), which was furnished by Google, and Giblin and Hada (2008) for a problem statement from IBM. More generally, "separation of duties" in management control refers to assigning the tasks and associated privileges for a specific business process across multiple functions with the primary objective of preventing fraud and errors (Botha and Eloff 2001). Giblin and Hada (2008) illustrate this as follows: "As a simple example, in a purchasing process, the person who requests a purchase usually is not the same person who approves purchases. Distributing responsibilities reduces the impact that a single individual can have, requiring collusion to perpetuate a fraud." This problem has recently received particular attention in the context of computerized (especially web-based) order processing (see Sun et al. 2010), and can be formalized as a parallel-machine scheduling problem with conflicts. Another related problem stems from Gaur et al. (2009), who schedule television commercials in program breaks, where insertion of competing commercials into the same break is undesirable. Balachandran and Zoltners (1981) explicitly mention the desirability of a constraint that "if an auditor participates in a certain audit engagement, then this auditor should/should not participate in a similar engagement," where an audit engagement is a set of audit tasks for the same client. One other application is resource assignment in workforce planning: Gardi (2009) describes how the assignment of a set of tasks with known start and end times to employees can be seen as a coloring problem, with each color representing an employee and with an interval graph as the conflict graph. Gardi restricts the number of tasks to be assigned to each employee, whereas we will minimize the maximum workload over all employees. Finally, in the context of the so-called "traveling purchaser problem," Manerba and Mansini (2015) describe that it may be possible that incompatible products cannot be loaded on the same vehicle, which has an interpretation very similar to our setting.

PMC was already studied by Bodlaender et al. (1994). They obtain a number of hardness and approximation results for specific graph types, but they do not develop an exact algorithm. Bodlaender et al. present approximation algorithms for the case where a k-coloring of the conflict graph is known a priori, with $k+1 \leq m$; the worst-case ratio depends only on k and when $\frac{m}{k}$ tends to infinity then the worst-case ratio tends to 2. They also prove that, unless P = NP, no approximation algorithm can improve upon the worst-case ratio of 2.

Informally, problem $P||C_{\max}$ can be seen as a "dual" to the bin packing problem (BPP), where the bin capacities correspond to the makespan and the number of bins corresponds to the number of parallel machines (see also Dell'Amico et al. 2008). A similar pairing can be observed between PMC and the bin packing problem with conflicts (BPPC), where the latter problem consists in packing items in a minimum number of bins of limited capacity while avoiding joint assignments of items that are in conflict. Clearly, BPPC generalizes both BPP and VCP. Problem BPPC has recently been studied from a computational point of view by a number of researchers, see Muritiba et al. (2010), Elhedhli et al. (2011) and Sadykov and Vanderbeck (2013). The currently best exact algorithm was developed by Sadykov and Vanderbeck (2013), who used their black-box branch-and-price (B&P) solver BaPCod, which relies on a generic branching scheme and certain primal heuristics, together with a specific pricing oracle.

Notice that PMC is intuitively harder than BPPC. From a VCP viewpoint, in PMC there is a hard constraint on the number of colors that can be used (namely m), while this number is variable in BPPC. As a result and contrary to PMC, in BPPC there is no feasibility problem. in PMC one has to assign jobs to m machines in such a way that conflicting jobs are assigned to different machines and such that the makespan is minimized. Consequently, for a given PMC instance we first need to verify whether there exists an m-coloring for the conflict graph: this is a priori not known for a given instance.

The remainder of this article is structured as follows. In Section 2 we describe two linear formulations for PMC, and our global algorithmic structure is sketched in Section 3; the details of the algorithm are developed in Sections 4 to 6. The results of a series of computational experiments are reported in Section 7, where we also test the algorithm on datasets of the classic problem $P||C_{\max}$ (with empty conflict graph). We conclude the article in Section 8.

2. Linear formulations for PMC

We first provide an intuitive linear formulation in Section 2.1, followed by a set-covering formulation in Section 2.2.

2.1. Intuitive formulation

We formulate a natural mixed-integer programming (MIP) model for PMC, as follows. For every job j and machine i we introduce a binary variable x_{ij} that is equal to 1 if job j is

assigned to machine i and 0 otherwise. We also introduce a real variable y that will equal the makespan of the schedule. A possible MIP model for PMC is then given by:

$$\text{minimize } y \tag{1a}$$

$$\text{subject to } \sum_{i \in M} x_{ij} = 1 \qquad \forall j \in J \tag{1b}$$

$$x_{ij} + x_{ij'} \leq 1 \qquad \forall \{j, j'\} \in E, \forall i \in M \tag{1c}$$

$$\sum_{j \in J} p_j x_{ij} \leq y \qquad \forall i \in M \tag{1d}$$

$$x_{ij} \in \{0, 1\} \qquad \forall j \in J, \forall i \in M \tag{1e}$$

$$y \in \mathbb{R}. \tag{1f}$$

The first set of constraints (1b) ensures that every job is scheduled on exactly one machine. Inequalities (1c) force conflicting jobs to be scheduled on different machines. Constraints (1d) guarantee that for each machine the makespan y is at least the total processing time consumed on that machine. Although correct, this formulation is quite unpractical. One reason for the high intractability of the formulation is the inherent symmetry: rearranging the indices of the machines leads to equivalent solutions. This has undesirable consequences in a branch-and-bound (B&B) scheme: the number of equivalent solutions is exponential in m and can lead to a lot of redundant work by a linear solver. One can resort to symmetry-breaking constraints (SBCs) for reducing the redundant work by a linear solver (see Berghman et al. 2014). The work of Dell'Amico et al. (2008) suggests that the following approach will perform better. Our findings also indicate (see Section 7 and the Appendix) that including SBCs does not significantly improve the performance of formulation (1).

Formulation (1) is almost of a form on which Dantzig-Wolfe decomposition can be applied (see Martin 1999). To elaborate this, we first rewrite (1) in matrix notation, as follows:

$$\text{minimize } y \tag{2a}$$

$$\text{subject to } \sum_{i \in M} \mathbb{I}_n x_i = e_n \tag{2b}$$

$$\begin{pmatrix} 0 \\ -y \end{pmatrix} + \begin{pmatrix} A \\ p \end{pmatrix} x_i \leq \begin{pmatrix} e_{|E|} \\ 0 \end{pmatrix} \qquad \forall i \in M \tag{2c}$$

$$x_i \in \{0, 1\}^n \qquad \forall i \in M \tag{2d}$$

$$y \in \mathbb{R}, \tag{2e}$$

where $x_i = (x_{i1}, \ldots, x_{in})'$ for each $i \in M$, $p = (p_1, \ldots, p_n)$, A is the edge-node incidence matrix of the graph $G = (J, E)$, $e_n = (1, \ldots, 1)' \in \{0,1\}^n$, $e_{|E|} = (1, \ldots, 1)' \in \{0,1\}^{|E|}$ and $\mathbb{I}_n$ is the unity matrix of size n.

In order to be able to apply Dantzig-Wolfe decomposition we switch to the decision variant of the optimization problem: we introduce an upper bound C on the value of the objective function, and we denote the resulting decision problem by $P(C, m)$, which is to determine whether there exists a feasible schedule without conflicts and with maximum makespan C on m machines. Variable y then disappears from the constraints and the resulting constraint matrix has a block-angular structure. PMC can now be solved by determining the smallest C for which $P(C, m)$ yields a "yes" answer; this will be achieved by a binary search algorithm, which will be described in Section 3. Since we work with identical machines, $P(C, m)$ can be reformulated as follows: is it possible to partition the job set J in at most m stable sets of $G = (J, E)$ such that each stable set corresponds to a machine that consumes at most C time units? This coincides with the decision variant of BPPC. Throughout this article, we will use this relationship between PMC and BPPC to develop an exact algorithm for PMC and we therefore introduce new notation that makes it easier to switch between the two problems: PMC and BPPC will be denoted by $P(\cdot, m)$ and $P(C, \cdot)$, respectively. The Dantzig-Wolfe decomposition can be seen here as a special case of variable redefinition as was presented in Vanderbeck (2000); this leads to a set-covering formulation for $P(C, \cdot)$.

2.2. Set-covering formulation

Let $\mathcal{S}_C$ be the set containing all the inclusion-maximal stable sets S of $G = (J, E)$ with $\sum_{j \in S} p_j \leq C$. We introduce a binary variable λ_S for each $S \in \mathcal{S}_C$ such that λ_S is equal to 1 if the stable set is chosen and 0 otherwise. The goal is to select a minimum number of stable sets of $G = (J, E)$ such that each job is contained in one machine schedule:

$$\text{minimize} \quad \sum_{S \in \mathcal{S}_C} \lambda_S \tag{3a}$$

$$\text{subject to} \quad \sum_{S \in \mathcal{S}_C : j \in S} \lambda_S \geq 1 \qquad \text{for each } j \in J \tag{3b}$$

$$\lambda_S \in \{0,1\} \qquad \text{for each } S \in \mathcal{S}_C \tag{3c}$$

Objective function (3a) minimizes the number of machines required, while constraints (3b) impose that every job has to be executed on a machine. The answer to $P(C, m)$ is "yes"

if and only if there exists a solution for the constraints (3b)–(3c) for which the value of the objective function (3a) is not greater than m. This set-covering model for $P(C,\cdot)$ was considered by Elhedhli et al. (2011), Muritiba et al. (2010) and Sadykov and Vanderbeck (2013), where exact algorithms were developed for $P(C,\cdot)$ based on B&P. A difference with our setting is that we only need to check whether a partition exists with at most m bins (machines) and we do not need the optimal objective value.

In what follows we denote the IP model (3) by $F(C,\cdot)$. This formulation has an exponential number of variables, so explicitly generating all these variables directly is impractical. Also, even if we list all the stable sets of G, then the LP relaxation of $F(C,\cdot)$ would still be very hard to solve, see for example Mehrotra and Trick (1996). We therefore solve the formulation with a B&P algorithm, i.e. at each node of a B&B search tree we solve the LP relaxation of $F(C,\cdot)$ by means of column generation (Gilmore and Gomory 1961).

3. Overall algorithmic structure for solving PMC

In this section we briefly outline our algorithm for PMC $\equiv P(\cdot,m)$, which is the essential contribution of this article. First a lower bound $L(\cdot,m)$ and an upper bound (heuristic solution) $U(\cdot,m)$ on the minimum makespan are computed. We provide more information on the lower bounds in Section 5.1 and on the upper bounds in Section 5.2. When the heuristics do not succeed in finding a feasible solution, we invoke a feasibility test in an attempt to recognize instances with empty solution space; this test is the subject of Section 6. Conversely, when a feasible solution is found with one of the heuristics of Section 5.2, then this solution is improved by means of local search; see Section 5.3.

If $L(\cdot,m)=U(\cdot,m)$ then an optimal solution has been found, otherwise we start a binary search to identify the optimal objective function (in line with Dell'Amico et al. 2008). In this search procedure, we iteratively verify whether a feasible schedule exists with makespan at most $C^*=\lfloor\frac{L(\cdot,m)+U(\cdot,m)}{2}\rfloor$; this verification is established by the B&P algorithm of Section 4. Let $L_F(C,\cdot)$ denote the optimal objective value of the LP relaxation of $F(C,\cdot)$ (a lower bound). If $L_F(C^*,\cdot)>m$ then we replace the lower bound $L(\cdot,m)$ by C^*+1. Otherwise, if the solution is integral then $U(\cdot,m)$ is replaced by the makespan of this solution (which is at most C^*), and if none of the previous two conditions holds then the B&P algorithm will branch and apply the same tests at lower levels of the search tree.

4. Branch and price

Below we describe all the ingredients of a B&P framework for $F(C,\cdot)$ that is used to answer the problem $P(C,m)$, so to evaluate whether a feasible schedule of makespan C exists given the processing times of the jobs and the conflict graph.

4.1. Column generation

The LP relaxation of $F(C,\cdot)$ is obtained by relaxing the integrality constraints, and the constraints $\lambda_S \leq 1$ can also removed because they are redundant (given a feasible solution with $\lambda_S > 1$ for some stable set S, one can simply set λ_S equal to 1). Thus, we solve the LP defined by the objective function (3a), the constraints (3b), and

$$\lambda_S \geq 0 \text{ for each } S \in \mathcal{S}_C. \tag{4}$$

The dual of this LP formulation is given by:

$$\text{maximize} \sum_{j\in J} \xi_j \tag{5a}$$

$$\text{subject to} \sum_{j\in S} \xi_j \leq 1 \qquad \text{for each } S \in \mathcal{S}_C, \tag{5b}$$

$$\xi_j \geq 0 \qquad \text{for each } j \in J, \tag{5c}$$

where $\xi_1,\ldots,\xi_n$ are the dual variables associated to the constraints (3b) and the constraints (5b) are associated to variables λ_S.

The LP relaxation of $F(C,\cdot)$ is solved with column generation. This entails iteratively solving the restricted master problem (RMP), which contains only a restricted number of columns, and the pricing problem, which determines whether there exists a column that can be added to improve the current solution. A column in this case equates with a bounded stable set with capacity C. At each iteration we check whether one of the constraints (5b) is violated. If no constraint is violated then we have obtained an optimal solution to the full LP relaxation; otherwise, we add to RMP a (subset of) λ variable(s) that correspond to violated constraints (5b). The pricing problem is the following: given a current dual solution ξ^*, does there exist a bounded stable set S of G for which $\sum_{j\in S} \xi_j^*$ is greater than 1? The LP relaxation is typically solved faster if one considers the constraints that are strongly violated; thus it is of interest to identify a bounded stable set S with most negative reduced cost. This can be modeled as follows:

$$\text{maximize} \sum_{j\in J} \xi_j^* z_j \tag{6a}$$

$$\text{subject to } \sum_{j\in J} p_j z_j \le C \tag{6b}$$

$$z_j + z_{j'} \le 1 \qquad \text{for each } \{j, j'\} \in E, \tag{6c}$$

$$z_j \in \{0, 1\} \qquad \text{for each } j \in J. \tag{6d}$$

Model (6) is an IP formulation for the knapsack problem with conflicts (KPC), which was studied by Hifi and Otmani (2012) and Pferschy and Schauer (2009), among others. KPC is clearly NP-hard on general graphs, because it reduces to the maximum weighted stable-set problem when $C \ge \sum_{j=1}^{n} p_j$. Several algorithms have been proposed in the literature for solving KPC, see for example Hifi and Otmani (2012), Pferschy and Schauer (2009) and Sadykov and Vanderbeck (2013); we solve the pricing problem with a dedicated algorithm that was presented in Sadykov and Vanderbeck (2013), with some minor modifications as described in Section 4.3. Denote by $\alpha(\xi^*)$ the optimal value of formulation (6). Clearly, for any ξ with $\xi_j \ge 0$ for all $j \in J$ and $\alpha(\xi) \le 1$, the value $\sum_{j\in J} \xi_j$ is a lower bound for $L_F(C, \cdot)$.

4.2. Numerically safe lower bound

Held et al. (2012) point out that LP solvers use floating-point representations for all numbers. Consequently, the dual variables ξ_j of the LP relaxation of $F(C, \cdot)$ computed by these solvers are inexact and so the condition $\alpha(\xi) > 1$ may be hard to assess. This can lead to premature termination or to endless loops. This problem was circumvented in Held et al. (2012) by introducing a numerically safe lower bound in exact integer arithmetic, which was used to calculate a lower bound for VCP. We will apply the same technique to solve our set-covering formulation.

We transform the dual variables ξ_j obtained from the LP solver to integers $\pi_j = \lfloor K\xi_j \rfloor$, i.e., we re-scale the dual variables by a scale factor K. As a result,

$$\xi_j - \frac{1}{K} < \frac{1}{K}\pi_j \le \xi_j. \tag{7}$$

This leads to a lower $\frac{n}{K}$-approximation of $\sum_{j\in J} \xi_j$:

$$\left(\sum_{j\in J} \xi_j\right) - \frac{n}{K} < \frac{1}{K}\sum_{j\in J} \pi_j \le \sum_{j\in J} \xi_j. \tag{8}$$

For the representation of integers and the choice of K we will follow the choices of Held et al. (2012). In the KPC problem (6), we replace the dual variables ξ_j (the profits) by the integers $\pi_j = \lfloor K\xi_j \rfloor$. We add new columns to RMP until $\alpha(\pi) \le K$; when this holds, the value $K^{-1}\sum_{j\in J} \pi_j \equiv \underline{L_F}(C, \cdot)$ is a "numerically safe" lower bound for $F_L(C, \cdot)$.

4.3. Solving KPC

Since the profits and weights of our KPC problem are integers, we can make some minor adjustments to the recursive enumeration procedure for KPC that was developed by Sadykov and Vanderbeck (2013). This procedure combines a classic B&B for the 0-1 knapsack problem with an enumeration algorithm for the maximum clique problem by Carraghan and Pardalos (1990). Both enumeration procedures follow a depth-first-search strategy. The dual bounds in the recursive enumeration algorithm of Sadykov and Vanderbeck (2013) are obtained by simply ignoring the conflicts between the "free" items, which are items that have not yet been fixed via branching decisions. Denote the set of all free vertices by F and the set of all vertices that have already been selected into the knapsack by S^1.

At each node of the B&B tree we calculate an upper bound via the continuous relaxation of the residual knapsack problem on set F, ignoring the conflict constraints:

$$\text{maximize} \quad \sum_{j \in F} \pi_j z_j \tag{9a}$$

$$\text{subject to} \quad \sum_{j \in F} p_j z_j \leq C - \sum_{j \in S^1} p_j \tag{9b}$$

$$0 \leq z_j \leq 1 \qquad \text{for each } j \in F. \tag{9c}$$

The upper bound in the recursive enumeration of Sadykov and Vanderbeck (2013) is the Dantzig bound for the knapsack problem (see for example Kellerer et al. 2004). In our case, we can use upper bounds that have been developed specifically for 0-1 knapsack problems with integer weights and profits; we use the bound of Martello and Toth (1977). The items of F are sorted according to their efficiency (ratio of profit per weight) and hence the upper bound for problem (9) can be found in $O(n)$ time using a greedy algorithm. The remainder of our knapsack algorithm has the same structure as that of Sadykov and Vanderbeck (2013).

4.4. Branching rule

At each node of the B&P tree we solve the LP relaxation of $F(C, \cdot)$ as described in Section 4.1. One of the following three cases will occur at every node in the tree:

1. if $\underline{L_F}(C, \cdot)$ exceeds m then the current node can be pruned immediately;
2. if $\underline{L_F}(C, \cdot)$ is less than or equal to m and each variable has an integral value, then the exploration of the search tree is halted and $U(\cdot, m)$ is set equal to the makespan of this solution (which is at most C);

3. if $\underline{L_F}(C,\cdot)$ is less than or equal to m and a λ variable is fractional, then we first attempt to construct a feasible solution with the primal heuristic that is described in Section 4.5; if this does not succeed, we branch and create two child nodes (otherwise the exploration is halted and $U(\cdot,m)$ is updated).

The branching strategy of our B&P algorithm is as follows. We select two items (jobs) j and j' with $\{j,j'\} \notin E$ and create two new BPPC instances. In the first instance, we enforce the two jobs j and j' to be on the same machine, by merging j and j' to one job with processing time $p_j + p_{j'}$, which is conflicting with each job that conflicted with either j or j'. In the second instance, we ensure that items j and j' are assigned to different color classes, by adding a conflict between the jobs j and j'. This branching strategy was first proposed in Zykov (1949) for graph coloring. Both child nodes inherit the valid stable sets from the parent node.

For the branching choice, we follow Held et al. (2012). For each pair $j,j' \in J$, define

$$q(j,j') = \frac{\sum_{S\in\mathcal{S}'_C : j,j'\in S} \lambda_S}{\frac{1}{2}(\sum_{S\in\mathcal{S}'_C : j\in S} \lambda_S + \sum_{S\in\mathcal{S}'_C : j'\in S} \lambda_S)},$$

where $\mathcal{S}'_C$ is set of all the current columns in the restricted master problem. It can be seen that $q(j,j') \in [0,1]$, and if $q(j,j')$ is close to 0 then the current solution assigns different fractional colors to j and j'. Conversely, if $q(j,j')$ is close to 1 then the two items are assigned to nearly equal fractional colors. In both of these cases, the lower bound in one child node upon branching on $\{j,j'\}$ will be similar to the lower bound in the parent node, hence we seek to branch on a pair $\{j,j'\}$ with $q(j,j')$ as close as possible to 0.5. The child node in which j and j' are assigned to the same machine is explored first; one reason is that this increases the probability of finding a feasible (integer) solution.

4.5. Primal heuristic

As mentioned before, we do not need to solve $P(C,\cdot)$ to optimality; we are only trying to find a partition of J in at most m bins (machines) of capacity C. A primal heuristic can be very useful in this process. Previous research of Muritiba et al. (2010), Elhedhli et al. (2011) and Sadykov and Vanderbeck (2013) has indicated that formulation $F(C,\cdot)$ is a very tight formulation for BPPC. Moreover, Sadykov and Vanderbeck (2013) show that this formulation combined with a column-generation-based primal heuristic can be very successful.

At every node of the B&P tree, before branching as described in Section 4.4, we first apply a generic diving heuristic that is a greedy heuristic search procedure, which was also used

in Sadykov and Vanderbeck (2013). Iteratively, we solve the LP relaxation of $F(C,\cdot)$ with column generation and then we create a smaller problem that results from rounding one of the λ variables to 1; this variable is selected greedily (fractional variable closest to 1). We then update the LP relaxation of $F(C,\cdot)$ by deleting the rows that correspond with the items that are covered by λ. We re-optimize the updated LP relaxation and repeat the process either until we find a partition of J in at most m bins (machines) of capacity C, or until the objective value of the updated LP relaxation exceeds m.

It frequently happens that the optimal value of the updated LP relaxation of $F(C,\cdot)$ is greater than the number of machines. It is therefore interesting to check whether fixing other variables would give better results, and thus further explore the solution space. We achieve such diversification by means of (limited) backtracking: we construct a different search tree for which the root node is equal to the B&P price node in which we have just finished computing the lower bound $\underline{L_F}(C,\cdot)$. This mechanism was developed by Joncour et al. (2010) and relies on the concept of limited discrepancy search (LDS) by Harvey and Ginsberg (1995). LDS essentially prevents the greedy strategy from choosing columns in a tabu list, which contains columns that were selected in previous branches. The tabu list at a node is the union of the tabu list of its ancestor and the columns chosen in previous child nodes of the ancestor. The tabu list at the beginning of the heuristic is empty. We explore a node that is not the first child of the ancestor if and only if the size of the tabu list is less than or equal to maxDiscrepancy and its depth does not exceed maxDepth.

5. Lower and upper bounds

In this section we briefly review some results from the literature on lower and upper bounds for $P||C_{\max}$ and VCP and extend some bounds to the case of PMC.

5.1. Lower bounds

Lower bounds for the parallel machine scheduling problem $P||C_{\max}$ are immediate lower bounds for PMC. Good lower bounds for $P||C_{\max}$ are:

$$L_0 = \left\lceil \frac{\sum_{j=1}^{n} p_j}{m} \right\rceil, \tag{10}$$

$$L_1 = \max\{L_0, p_1\}, \tag{11}$$

$$L_2 = \max\{L_1, p_m + p_{m+1}\}. \tag{12}$$

Dell'Amico and Martello (1995) prove that the worst-case performance ratio for L_2 is equal to $\frac{2}{3}$ for $P||C_{\max}$. This does not necessarily carry over to PMC, however, because for a given instance it is not even sure whether there exists a feasible solution. The lower bounds (10)–(12) have the advantage of requiring low computation time, namely $O(n)$.

One can also construct lower bounds that are tailored to the PMC structure, using the relation between PMC and BPPC. We describe a tight lower bound for $P(\cdot, m)$ with the help of the LP relaxation of $F(C, \cdot)$. Suppose that we have constructed a feasible solution for $P(\cdot, m)$, then set a variable U to be equal to the makespan of this feasible solution, and another variable $L := L_2$. In a binary search we check whether the optimal value of the LP relaxation of $F(C, \cdot)$ with $C = \frac{L+U}{2}$ is greater than m. If the answer to this question is "yes," then we can set L equal to $C+1$, otherwise we set U equal to C. We repeat this while $L < U$. Lower bound L_3 is the final value of L; this will be a tight bound, but finding it is quite time-consuming. We invoke this procedure only if the constructive heuristics of Section 5.2 have already produced five feasible solutions but none these had a makespan equal to the current lower bound.

5.2. Upper bounds

Approximation algorithms for $P||C_{\max}$ cannot be used directly as upper bounds for PMC because these algorithms do not consider the conflict graph, so we will extend such algorithms to apply for PMC. One of the best-known approximation algorithms for for $P||C_{\max}$ is the LPT (*longest processing time*) algorithm of Graham (1966, 1969). This algorithm first orders the jobs by non-increasing processing times and then iteratively assigns each job to a machine with lowest current maximum completion time. Obviously, due to the conflict graph it may not always possible to assign a job to the selected machine; in this case we assign the job to a machine with lowest maximum completion time without conflicting jobs. This procedure might still break down, however, in case every machine already contains a conflicting job. If this occurs then the algorithm is interrupted, the jobs are ordered randomly and the procedure is restarted. We iterate this procedure until a given number of feasible solutions is found, or until no feasible solution is found for a given number of iterations (since the instance can be infeasible).

Bodlaender et al. (1994) propose a general heuristic for PMC in the case that we know some k-coloring a priori for the given conflict graph, with $k < m$. The makespan produced by this algorithm is bounded by a constant that depends on k and m, multiplied with the optimum

makespan. We briefly describe one of these heuristics; the other heuristics are similar and depend on the relation between k and m (and they were all implemented). Suppose that the conflict graph has a k-coloring such that $m > 2(k-1)$. The algorithm assigns to the k color classes $C_1, \ldots, C_k$ of the conflict graph disjoint sets of μ_i machines, with $1 \leq i \leq k$. Denote by P_i the sum of the processing times of all jobs assigned to color class C_i and assign to each color class $\mu_i = \left\lceil \frac{P_i}{2L_1} \right\rceil$ machines. Bodlaender et al. show that $\sum_{i=1}^{k} \mu_i \leq m$. Next they assign each of the jobs of C_i to one of the μ_i machines of the color class using the LPT algorithm of Graham. This heuristic has a worst-case performance ratio of $3 - \frac{1}{m-k+1}$. In preliminary experiments we have found that $\sum_{i=1}^{k} \mu_i$ is often strictly smaller than m. We therefore slightly modify the algorithm, as follows: we iteratively assign the remaining machines to the color class C_i for which $\frac{P_i}{\mu_i}$ is maximal, until all the machines are assigned.

One can only use the algorithm of Bodlaender et al. (1994) when a k-coloring of the conflict graph is given and hence we have to construct k-colorings such that $k \leq m$, which is of course not always possible. We use the standard greedy algorithms SEQ and DSATUR (see Johnson et al. (1991) and Brélaz (1979), respectively). SEQ is a simple greedy algorithm for the VCP. We order the nodes of the conflict graph randomly, and we assign the first node to the first color class, the second node in the list is assigned to the first color class that contains no nodes that are adjacent to that node, and so on. DSATUR is very similar to SEQ, but DSATUR chooses dynamically which node to color first: at each step it assigns the node that is adjacent to the largest number of distinctly different colored nodes (after a few randomly colored nodes).

5.3. Local search

Every solution that is obtained by one of the heuristics described in Section 5.2 is improved with a $(k-l)$-swap procedure. This procedure swaps groups of jobs between two machines. Again, we should obviously take the conflict graph into account and check whether a swap is allowed. Our description is an extension of the procedure of Dell'Amico et al. (2008). Consider two machines m_1 and m_2, denote the current completion times of these machines by $C(m_1)$ and $C(m_2)$ respectively, and let M_1 and M_2 be the sets of jobs that are currently assigned to each of the machines. The swap procedure aims to exchange k jobs currently assigned to machine m_1 and gathered in set K, with l jobs on m_2 and gathered in set L, such that the resulting value $\max\{C(m_1), C(m_2)\}$ decreases and such that $M_2 \setminus L$ does not contain any job that conflicts with the k jobs of K, and vice versa.

The procedure is started by creating two lists of machines. The first list contains the machines for which the completion time is greater than lower bound L_3; this list is sorted by non-increasing maximum completion time. The second list contains all other machines and is sorted by non-decreasing maximum completion time. For a subset Q of jobs, define $P(Q) = \sum_{j \in Q} p_j$. For each machine m_1 of the first list, in order, consider the subset $K \subset M_1$ of k jobs such that $P(K)$ is maximal and execute the following steps:

1. find a machine m_2 in the second list, if any, with a subset $L \subset M_2$ with l jobs such that $P(L) < P(K)$, $C(m_2) - P(L) + P(M) < C(m_1)$, $M_2 \setminus L$ does not contain any job that is in conflict with the jobs of K and vice versa, and interchange the sets K and L;

2. if a machine with the above-mentioned properties is not found then take the next largest subset K of M_1, if any, and go to Step 1.

As soon as a feasible exchange is found, it is performed, and the procedure is restarted from the obtained solution until no feasible exchange is found.

6. Feasibility tests

As noticed earlier, it will be not always possible to quickly construct a k-coloring for a given conflict graph $G = (J, E)$ such that $k \leq m$, or to quickly obtain a feasible solution for a given PMC instance. The chromatic number of a graph is the lowest number of colors needed to color the vertices so that no two adjacent vertices share the same color. Obviously, if the chromatic number of G is greater than the number of the machines m then the instance is infeasible.

We examine a number of lower bounds for the chromatic number of the conflict graph. A clique is a set of vertices such that any pair of vertices is adjacent. The clique number of a graph is the maximum size of a clique in G. Clearly, the size of an arbitrary clique as well as the clique number are both lower bounds for the chromatic number, because in any clique all the vertices require different colors. A major disadvantage here is that the worst-case ratio between the clique number and the chromatic number is arbitrarily bad (see for instance Hastad 1996) and finding a clique of maximum size is also an NP-hard problem (Garey and Johnson 1979). Johnson (1973) describes an efficient heuristic for the maximum clique problem, however, and Östergård (2001) proposes an efficient exact algorithm. We use these algorithms to quickly find a lower bound for the chromatic number. As soon as the constructed lower bound exceeds m then we stop and conclude that the instance is infeasible because the conflict graph is overly restrictive.

We describe another effective and tighter lower bound, which was first considered in Mehrotra and Trick (1996), who propose a set-covering model for VCP that is similar in nature to model $F(C,\cdot)$. Let $\mathcal{S}$ be the set containing all the inclusion-maximal stable sets S of $G=(J,E)$. With each stable set S we associate a binary variable λ_S that is equal to 1 if and only if stable set S is chosen. VCP can be modeled as follows:

$$\text{minimize} \sum_{S\in\mathcal{S}} \lambda_S \tag{13a}$$

$$\text{subject to} \sum_{S\in\mathcal{S}:j\in S} \lambda_S \geq 1 \qquad \text{for each } j\in J \tag{13b}$$

$$\lambda_S \in \{0,1\} \qquad \text{for each } S\in\mathcal{S} \tag{13c}$$

Objective function (13a) minimizes the number of colors required for allowing a feasible solution. Constraints (13b) impose that every item (job) has to be colored. The constraints (13c) require variables λ_S to be binary. Note that if an item j belongs to more than one color class (machine), it can be removed from all but one of these classes, which leads to a feasible solution with the same objective. Relaxing the integrality constraints (13c) gives the LP relaxation of the set-covering model for VCP, in which as before we can disregard the constraints $\lambda_S \leq 1$. Denote by z^* an optimal solution to this LP relaxation; $\lceil z^* \rceil$ is a valid lower bound for the VCP. This model also has an exponential number of variables and hence the LP relaxation of (13) is solved with column generation, as before. The only difference is the pricing problem, which is here a maximum weighted stable set problem, in other words, there is no capacity constraint such as (6b). We solve the pricing problem with Algorithm 1 of Held et al. (2012). We branch until the algorithm finds a k-coloring with $k\leq m$, or until we have shown that there is no feasible m-coloring for the conflict graph. The further algorithmic details of the B&P algorithm are similar to the B&P algorithm for $F(C,\cdot)$. The primal heuristic of Section 4.5 is used also here in order to quickly find a feasible solution.

7. Computational experiments

In this section we provide details about our experimental setup (Section 7.1), and we report computational results on datasets containing instances without conflict graph (meaning an empty graph; in Section 7.2) and with arbitrary conflict graph (Section 7.3). In Section 7.4 we examine the behavior of our algorithm as a function of some of the problem's parameters in more detail, and finally thorough comparisons are made with the performance of a standalone linear solver on a compact formulation of PMC in Section 7.5.

7.1. Experimental setup

The algorithms have been implemented in the C programming language and compiled with gcc version 4.8.2 with full optimization pack -O3. The computational experiments have been performed on one core of a system with Intel Core i7-3770 processor at 3.4 GHz and 8 GB of RAM under a Linux OS. We use the following experimental settings for our algorithm. In the initialization phase, we construct 20 different solutions if this possible; we stop if for n iterations there is no new or feasible solution. For the local search $(k-l)$-swap procedure we choose $k \in \{1,2\}$ and $l \in \{0,1,2\}$; all these swaps are called in a random order. If the pricing problem is a KPC problem with $d > 0.1$ (see below for definition of d) then it is solved with the algorithm of Section 4.3, otherwise the problem is solved by the general MIP solver Gurobi 6.0.0. If the pricing problem is a 0/1 knapsack problem then we call Combo (Martello et al. 1999), which is publicly available from `http://www.diku.dk/pisinger/codes.html`. All LPs are solved with Gurobi. In the primal heuristic we set maxDiscrepancy $= 2$ and maxDepth $= 3$. Each run of the algorithm (for one instance) is interrupted after 900 seconds.

The algorithms have been experimentally tested on a large set of PMC instances that were randomly generated as follows. The number n of jobs is either $10, 25, 50, 75$ or 100. The processing times are integers randomly generated from a uniform distribution in a given range $[a,b]$; we consider the ranges $[1,10], [1,50]$ and $[1,100]$. The conflict graphs are generated using the Networkx module in Python. The algorithm chooses each of the $\frac{n(n-1)}{2}$ possible edges with probability d; we consider values $d = 0.1, 0.2,\ 0.3, 0.4, 0.5$. As a result, we obtain triples (n,d,b) with $n \in \{10,25,50,75,100\}$, $d \in \{0.1,0.2,0.3,0.4,0.5\}$ and $b \in \{10,50,100\}$. For each triple we generate 10 instances. We schedule these instances on $m = 5, 10, 15$ and 20 machines.

We have also tested our algorithms on instances of $P||C_{\max}$, in which there are no conflicting jobs. These instances were tested in Dell'Amico et al. (2008), and they consist of two main groups, referred to as "uniform" and "nonuniform." The uniform instances have processing times drawn from a uniform distribution in a range $[a,b]$ and were first proposed by França et al. (1994). The nonuniform instances are obtained by randomly generating 98% of the processing times from a uniform distribution on $[0.9(b-a), b]$ and the remaining processing times using a uniform distribution on $[a, 0.2(b-a)]$; they were generated in Frangioni et al. (2004). Both the uniform and the nonuniform instance sets contain three different classes, corresponding to different intervals for the processing times: $a = 1$ in all three classes but b

is either 100, 1000 or 10000. Each such class consists of 13 pairs (m, n), with $m \in \{5, 10, 25\}$, $n \in \{10, 25, 100, 500, 1000\}$ and $m < n$; there are 10 instances for each pair.

7.2. Computational results on instances without conflicts

We first compare with heuristic procedures (Section 7.2.1) and afterwards with exact procedures for $P||C_{\max}$ (Section 7.2.2).

7.2.1. Heuristics for $P||C_{\max}$

We first examine the performance of the part of our exact algorithm that consists of the constructive approximation algorithms of Section 5.2, the local search improvements of Section 5.3 for every constructed solution, the lower bounds of Section 5.1, and the primal heuristic of Section 4.5 at the root node of the B&P tree for the makespan value equal to the best lower bound. In other words, we test whether the primal heuristic can find a feasible solution for a BPP instance at the root node. We call this (heuristic) part of our exact algorithm KL-heur. We compare with two of the best performing (meta-)heuristic procedures from the literature: AR (Alvim and Ribeiro 2004), which is a binary search algorithm that invokes tabu search at each iteration to try to find a feasible solution for a BPP instance, and DIMM-SS (Dell'Amico et al. 2008), which is a meta-heuristic algorithm based on the scatter search paradigm that optimally solves many benchmark instances.

For each group of instances (uniform and nonuniform), and for each of the three algorithms, Table 1 reports the number of optimal solutions (#opt) and the average CPU time (sec) in seconds. Each regular cell in the table contains the values corresponding with 10 instances of a given class that depends on the number of jobs (n), the number of machines (m) and the group (uniform or nonuniform). For each group and each range of processing times we compute also the *average* over the 130 instances. The *overall average* is the average over 390 instances of each group. It is to be noted that algorithm AR was run on a machine with a AMD 2.4 GHz processor, and DIMM-SS was tested on a machine with a Pentium IV 3.0 GHz processor; the results for the algorithms were taken from Dell'Amico et al. (2008).

Contrary to AR and DIMM-SS, our algorithm KL-heur solves all the instances to guaranteed optimality. With 0.168 CPU seconds average runtime across all instances, algorithm KL-heur is also faster than the algorithms DIMM-SS (2.77 CPU seconds) and AR (0.175 CPU seconds), but the average CPU time for KL-heur on some subsets of nonuniform instances is clearly far higher than the average CPU time for AR and DIMM-SS. This probably stems

Table 1 Heuristic algorithms: uniform and nonuniform instances

			uniform						nonuniform					
			AR		DIMM-SS		KL-heur		AR		DIMM-SS		KL-heur	
range	m	n	sec	#opt	sec	#opt	sec	#opt	sec	#opt	sec	#opt	sec	#opt
$[1, 10^2]$	5	10	0.00	10	0.00	10	0.00	10	0.00	10	0.00	10	0.00	10
	5	50	0.00	10	0.00	10	0.00	10	0.03	10	0.00	10	0.11	10
	5	100	0.00	10	0.00	10	0.00	10	0.01	10	0.00	10	0.00	10
	5	500	0.00	10	0.00	10	0.00	10	0.01	10	0.04	10	0.00	10
	5	1000	0.00	10	0.00	10	0.00	10	0.02	10	0.17	10	0.01	10
	10	50	0.00	10	0.00	10	0.00	10	0.31	4	0.00	10	0.04	10
	10	100	0.00	10	0.00	10	0.00	10	0.22	8	0.00	10	0.43	10
	10	500	0.00	10	0.01	10	0.00	10	0.00	10	0.03	10	0.00	10
	10	1000	0.00	10	0.00	10	0.01	10	0.00	10	0.11	10	0.00	10
	25	50	0.01	9	3.00	9	0.01	10	0.00	10	0.00	10	0.00	10
	25	100	0.00	10	0.00	10	0.00	10	0.55	8	0.00	10	0.10	10
	25	500	0.00	10	0.00	10	0.00	10	0.08	10	0.46	10	0.00	10
	25	1000	0.00	10	0.00	10	0.01	10	0.18	10	1.02	10	0.01	10
average			0.00	129	0.23	129	0.00	130	0.11	120	0.14	130	0.05	130
$[1, 10^3]$	5	10	0.00	10	0.00	10	0.00	10	0.00	10	0.00	10	0.00	10
	5	50	0.00	10	0.00	10	0.00	10	0.03	10	0.00	10	0.34	10
	5	100	0.00	10	0.00	10	0.00	10	0.02	10	0.00	10	0.00	10
	5	500	0.02	10	0.03	10	0.00	10	0.00	10	0.03	10	0.00	10
	5	1000	0.05	10	0.11	10	0.04	10	0.02	10	0.18	10	0.01	10
	10	50	0.02	8	0.01	10	0.08	10	0.00	10	0.00	10	0.14	10
	10	100	0.00	10	0.00	10	0.00	10	0.35	10	0.02	10	1.24	10
	10	500	0.01	10	0.02	10	0.00	10	0.07	10	0.03	10	0.00	10
	10	1000	0.04	10	0.11	10	0.02	10	0.06	10	0.18	10	0.01	10
	25	50	0.02	9	3.00	9	0.02	10	0.00	10	0.00	10	0.00	10
	25	100	0.04	8	20.34	9	0.50	10	1.08	8	0.02	10	0.41	10
	25	500	0.00	10	0.02	10	0.00	10	0.13	10	0.72	10	0.00	10
	25	1000	0.02	10	0.09	10	0.01	10	0.43	10	0.43	10	0.01	10
average			0.02	125	1.82	128	0.05	130	0.17	128	0.12	130	0.17	130
$[1, 10^4]$	5	10	0.03	8	0.00	9	0.00	10	0.00	10	0.00	10	0.00	10
	5	50	0.03	10	0.00	10	0.02	10	0.02	10	0.00	10	1.10	10
	5	100	0.00	10	0.00	10	0.00	10	0.01	10	0.02	10	0.01	10
	5	500	0.01	10	0.03	10	0.03	10	0.00	10	3.00	10	0.00	10
	5	1000	0.14	10	0.21	10	0.01	10	0.03	10	0.18	10	0.04	10
	10	50	0.25	0	18.37	6	0.39	10	0.35	8	0.01	10	0.48	10
	10	100	0.00	10	0.01	10	0.05	10	0.24	10	0.03	10	4.36	10
	10	500	0.01	10	0.02	10	0.02	10	0.01	10	0.05	10	0.01	10
	10	1000	0.04	10	0.15	10	0.02	10	0.02	10	0.12	10	0.01	10
	25	50	0.03	9	3.00	9	0.00	10	0.00	10	0.00	10	0.01	10
	25	100	0.59	0	120.03	0	1.33	10	4.29	3	39.82	7	1.55	10
	25	500	0.01	10	0.02	10	0.03	10	3.13	10	1.23	10	0.03	10
	25	1000	0.03	10	0.09	10	0.04	10	0.22	10	2.75	10	0.03	10
average			0.09	107	10.92	114	0.15	130	0.64	121	3.40	127	0.59	130
overall average			0.04	361	4.32	371	0.07	390	0.31	369	1.22	387	0.27	390

AR: AMD 2.4 GHz, DIMM-SS: Pentium IV 3 GHz, KL-heur: Intel Core i7-3770 3.4 GHz

from the calculation of the lower bound for these instances: we calculate a tight lower bound with the help of the set covering relaxation $F(C, \cdot)$ of BPP if the constructive heuristics do not succeed in confirming optimality of the given lower bound. It can be anticipated that for larger and/or more difficult instances, iteratively calling the primal heuristic more than once might lead to better solutions, but this turns out not to be necessary here.

7.2.2. Exact algorithms for $P||C_{\max}$

Next we compare our overall algorithm with the best exact algorithms for $P||C_{\max}$ from the literature, namely with DM (Dell'Amico and Martello 1995) and with DIMM (Dell'Amico et al. 2008). DM is a branch-and-bound algorithm that computes a lower bound at every node, based on the relation between $P||C_{\max}$ and BPP. Some dominance criteria are applied, and the initialization phase of the algorithm consists of a number of approximation algorithms for $P||C_{\max}$ (drawn from the literature). DIMM is an algorithm that consists of two phases. The first phase is the scatter search algorithm DIMM-SS, which, as mentioned in Section 7.2.1, already solves many instances to optimality. The second phase of the algorithm is based on a binary search and a B&P scheme.

We refer to our own algorithm as KL. For the two groups of instances and for each range of processing times, Table 2 contains the average CPU time (sec), average percentage gap (%gap) and the number of optimal solutions found (#opt). We provide the same information for both groups of instances, uniform and nonuniform, and for all 780 instances. The algorithms DIMM and KL clearly outperform algorithm DM, in that DM does not produce optimal solutions for all instances despite the higher average runtimes. Also, the average CPU time for KL is clearly lower than the average CPU time of DM.

Table 2 **Exact algorithms for $P||C_{\max}$**

		DM			DIMM		KL	
class	range	%gap	sec	#opt	sec	#opt	sec	#opt
uniform	$[1, 10^2]$	0.0000	0.00	130	0.23	130	0.00	130
	$[1, 10^3]$	0.0019	21.75	127	1.86	130	0.05	130
	$[1, 10^4]$	0.0348	146.98	109	13.5	130	0.15	130
average		0.0122	56.25	366	5.19	390	0.07	390
nonuniform	$[1, 10^2]$	0.0155	131.56	120	0.14	130	0.05	130
	$[1, 10^3]$	0.0169	124.66	112	0.12	130	0.17	130
	$[1, 10^4]$	0.0190	346.32	80	10.99	130	0.59	130
average		0.0171	200.85	312	3.75	390	0.27	390
overall average		0.0147	128.55	678	4.47	780	0.17	780

DM, DIMM: Pentium IV 3 GHz, KL: Intel Core i7-3770 3.4 GHz

7.3. Computational results on instances with conflicts

In Table 3 we report for each range of processing times, number of machines and number of jobs the average CPU time (sec) in seconds, the average percentage gap (%gap) and the number of optimal solutions (#opt), where this last value counts the number of instances for which we have found a guaranteed optimal solution or proved that the instance is infeasible. Each regular cell in the table pertains to 50 instances (10 for each value of $d \in \{0.1, 0.2, 0.3, 0.4, 0.5\}$). We see that the algorithm solves all the instances if the number of jobs is at most 50. Unsolved instances (with 900 seconds runtime limit) occur with 75 and 100 jobs; the highest runtimes and highest numbers of unsolved instances are found when the instances "balance" between feasible and infeasible, meaning that the chromatic number of the conflict graph is close to m. Indeed, in this case it is time-consuming to prove that an instance is unfeasible, and the feasible instances are quite constrained due to the high density of the graph, or put differently: this is when the algorithm needs time to decide whether an instance is feasible or unfeasible. If the number of machines is sufficiently low then it will be easier to decide that an instance is infeasible, ceteris paribus. For instances with 75 jobs we observe that a transition (from feasible to infeasible) takes place for 10 machines. The same can be observed for instances with 100 jobs, but then the transition point can be at $5, 10, 15$ or 20 machines, dependent on the density of the instance. Overall, we solve all instances in the dataset in an average runtime of 44.721 seconds, with an average optimality gap of 0.276% and we find a guaranteed optimal solution to 2436 out of the 2550 instances in total (these aggregate figures are not very informative, however).

Table 4 displays the same computational results but now gathered per value of the density of the conflict graph, the number of machines and the number of jobs. Each class has 30 instances in this table. For $n = 100$, for example, it can be clearly seen that the most difficult instances have $m = 5$ when $d = 0.1$, $m = 10$ when $d = 0.3$, $m = 15$ when $d = 0.4$ and $m = 15$ and 20 when $d = 0.5$. The absence of "difficult" instances for density $d = 0.2$ is easily explained: the hard values for m are especially 7 and 8 here (which we have experimentally confirmed, but we do not report the values here for brevity of exposition). As explained above, these distinguishing values for m will typically coincide with the chromatic number of the conflict graph.

Table 3 Computational results on instances with conflicts in function of range

(a) Range $[1, 10]$

m	n	sec	%gap	#opt
5	10	0.001	0.000	50
20	25	0.000	0.000	50
15	25	0.001	0.000	50
10	25	0.002	0.000	50
5	25	0.025	0.000	50
20	50	0.005	0.000	50
15	50	0.008	0.000	50
10	50	0.385	0.000	50
5	50	3.423	0.000	50
20	75	0.037	0.000	50
15	75	0.193	0.000	50
10	75	162.632	1.068	41
5	75	0.334	0.000	50
20	100	0.242	0.000	50
15	100	180.686	1.235	40
10	100	100.781	0.221	45
5	100	52.016	0.246	49
average		29.457	0.281	825

(b) Range $[1, 50]$

m	n	sec	%gap	#opt
5	10	0.002	0.000	50
20	25	0.000	0.000	50
15	25	0.001	0.000	50
10	25	0.030	0.000	50
5	25	0.055	0.000	50
20	50	0.107	0.000	50
15	50	0.130	0.000	50
10	50	1.455	0.000	50
5	50	21.564	0.000	50
20	75	0.251	0.000	50
15	75	14.055	0.000	50
10	75	182.456	2.284	40
5	75	0.826	0.000	50
20	100	1.327	0.000	50
15	100	221.026	1.530	38
10	100	181.440	0.413	40
5	100	165.632	0.186	42
average		46.492	0.260	810

(c) Range $[1, 100]$

m	n	sec	%gap	#opt
5	10	0.002	0.000	50
20	25	0.000	0.000	50
15	25	0.003	0.000	50
10	25	0.046	0.000	50
5	25	0.067	0.000	50
20	50	0.117	0.000	50
15	50	0.307	0.000	50
10	50	2.618	0.000	50
5	50	45.341	0.000	50
20	75	0.702	0.000	50
15	75	78.777	0.022	48
10	75	201.388	2.505	39
5	75	2.284	0.000	50
20	100	63.991	0.024	47
15	100	244.608	1.701	37
10	100	164.637	0.407	40
5	100	184.755	0.214	40
average		58.214	0.287	801

7.4. Phase transitions

It was suggested above that an easy-hard-easy transition occurs, dependent on the values of some of the parameters. We now examine this behavior in more detail. Figure 1 displays the average CPU time (on a logarithmic scale) for solving 10 instances as a function of the density of the conflict graph (which is on the horizontal axis); the different curves correspond with different numbers m of machines. The instances were created as follows: each instance has

Table 4 Computational results on instances with conflicts in function of density

(a) $d = 0.1$

m	n	sec	%gap	#opt
5	10	0.001	0.000	30
20	25	0.000	0.000	30
15	25	0.000	0.000	30
10	25	0.026	0.000	30
5	25	0.001	0.000	30
20	50	0.161	0.000	30
15	50	0.071	0.000	30
10	50	0.011	0.000	30
5	50	0.014	0.000	30
20	75	0.039	0.000	30
15	75	0.001	0.000	30
10	75	0.003	0.000	30
5	75	4.185	0.000	30
20	100	0.002	0.000	30
15	100	0.001	0.000	30
10	100	0.002	0.000	30
5	100	653.826	1.077	11
average		38.726	0.063	491

(b) $d = 0.2$

m	n	sec	%gap	#opt
5	10	0.001	0.000	30
20	25	0.000	0.000	30
15	25	0.001	0.000	30
10	25	0.012	0.000	30
5	25	0.055	0.000	30
20	50	0.026	0.000	30
15	50	0.035	0.000	30
10	50	0.052	0.000	30
5	50	117.112	0.000	30
20	75	0.064	0.000	30
15	75	0.140	0.000	30
10	75	0.406	0.000	30
5	75	1.376	0.000	30
20	100	0.232	0.000	30
15	100	0.343	0.000	30
10	100	5.280	0.000	30
5	100	16.781	0.000	30
average		8.348	0.000	510

(c) $d = 0.3$

m	n	sec	%gap	#opt
5	10	0.002	0.000	30
20	25	0.000	0.000	30
15	25	0.001	0.000	30
10	25	0.021	0.000	30
5	25	0.086	0.000	30
20	50	0.066	0.000	30
15	50	0.095	0.000	30
10	50	0.243	0.000	30
5	50	0.077	0.000	30
20	75	0.165	0.000	30
15	75	0.392	0.000	30
10	75	36.370	0.008	29
5	75	0.159	0.000	30
20	100	0.706	0.000	30
15	100	2.142	0.000	30
10	100	737.994	1.735	5
5	100	0.015	0.000	30
average		45.796	0.103	484

(d) $d = 0.4$

m	n	sec	%gap	#opt
5	10	0.002	0.000	30
20	25	0.000	0.000	30
15	25	0.005	0.000	30
10	25	0.031	0.000	30
5	25	0.103	0.000	30
20	50	0.042	0.000	30
15	50	0.170	0.000	30
10	50	2.379	0.000	30
5	50	0.006	0.000	30
20	75	0.406	0.000	30
15	75	1.192	0.000	30
10	75	873.899	9.754	1
5	75	0.009	0.000	30
20	100	1.898	0.000	30
15	100	174.292	0.069	25
10	100	1.057	0.000	30
5	100	0.021	0.000	30
average		62.089	0.578	476

(e) $d = 0.5$

m	n	sec	%gap	#opt
5	10	0.002	0.000	30
20	25	0.000	0.000	30
15	25	0.002	0.000	30
10	25	0.040	0.000	30
5	25	0.001	0.000	30
20	50	0.086	0.000	30
15	50	0.371	0.000	30
10	50	4.744	0.000	30
5	50	0.004	0.000	30
20	75	0.977	0.000	30
15	75	153.315	0.036	28
10	75	0.117	0.000	30
5	75	0.011	0.000	30
20	100	106.429	0.040	27
15	100	900.422	7.375	0
10	100	0.431	0.000	30
5	100	0.028	0.000	30
average		68.646	0.438	475

50 jobs with processing times drawn from interval $[1, 100]$. Starting with an empty conflict graph, we stepwise randomly add edges until the density of the graph has increased by at least 5 percent, and this until the density of the graph reaches at least 90 percent. In this way we obtain 190 instances, with 10 instances for each value in $\{0.0, 0.05, 0.10, \ldots, 0.90\}$.

The peak average runtime occurs for higher densities as the number of machines rises; for $m = 5$, for instance, the highest runtimes are observed when $d = 0.20$. These instances are

Figure 1 **Average CPU time on a logarithmic scale (smoothed curve)**

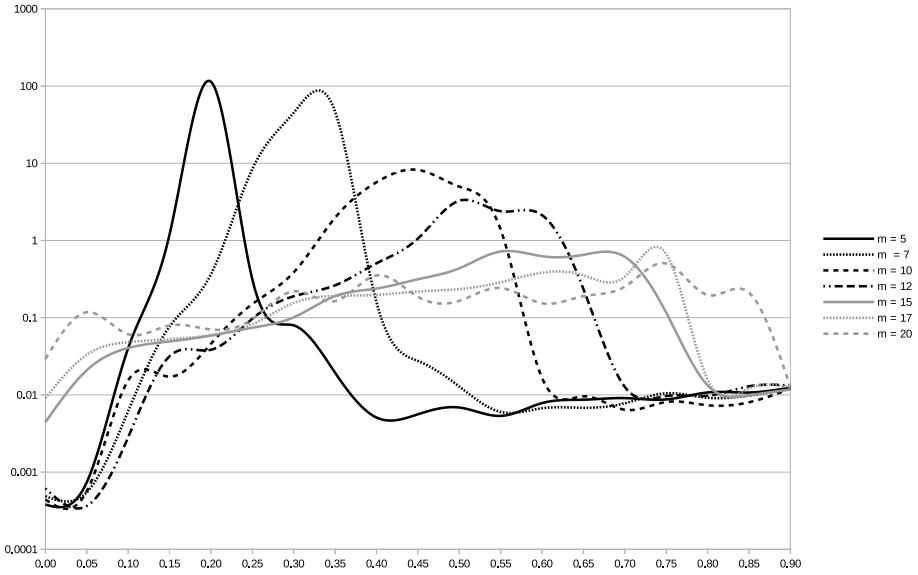
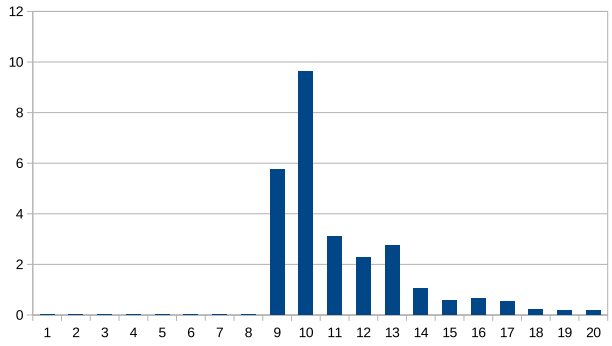


Figure 2 **CPU time for one instance as a function of the number of machines**



hard to color and the lower bound of Section 5.1 is not very tight anymore. As m goes up, the curves become lower and also flatter, indicating that there is less variability in the empirical hardness of instances. For densities above those with the highest runtime, the instances are typically infeasible, but the runtimes are still not always drastically lower because the algorithm sometimes calculates a lower bound on the chromatic number with the help of the formulation (13) after it has tried to find a clique with sufficient jobs. When the density becomes large enough then the construction of a clique will suffice to show infeasibility.

We have also looked into the effect on runtimes of the number of machines for a given graph. For one instance of our dataset with $n = 50$, $b = 100$ and $d = 0.5$ (the first instance with these settings, instance name pmc50_0.5_100_0.txt), Figure 2 shows the CPU time needed with different m-values (CPU time is on the vertical axis, m on the horizontal axis). The

first feasible solution is obtained with 9 machines; for m close to but less than 9 the runtime to show infeasibility is higher than for lower m because the construction of the clique takes longer or we have to compute a lower bound for the number of colors with the help of formulation (13), which can be time-consuming. The algorithm needs more time for $m = 10$ than for $m = 9$ because some parts of the search tree will require more effort to confirm infeasible makespan values. Beyond $m = 10$, it becomes easier to find an optimal solution.

7.5. Comparison with Gurobi

Since this is the first paper in which an exact algorithm for PMC is developed, we turn to the MIP solver Gurobi 6.0.0 with the compact formulation (1) for a benchmark comparison. Contrary to the foregoing sections, Gurobi uses four cores here. We have also tested this formulation with various SBCs, but without obtaining better solutions. The computational results for the extensions with SBCs can be consulted in the Appendix.

For Gurobi, Table 5 contains similar information as Table 3; we include an extra column that counts the number of instances (#no) for which the MIP solver could not decide whether the instance was feasible or infeasible after 900 seconds runtime. The average CPU time in every cell is computed over all the instances and the average gap in every cell is computed over all the instances for which we have at least one feasible solution or have proved that the instance is infeasible.

We observe that the MIP solver already experiences problems when $n = 50$ and with a high number m of machines, but that it gives very good results for instances with low m, so instances with a high number of jobs per machine. In such instances, the number of equivalent solutions is much smaller and symmetry is not a large issue for the MIP solver. Gurobi is better than our algorithm for the setting $m = 5$, $n = 100$, which is the lowest line in each of the three subtables of Table 5: it is faster and solves more instances to optimality. For all other settings, our algorithm is dominant. Our difficulty with $m = 5$ and $n = 100$ can be explained by the fact that the lower bound of Section 5.1 is not very tight anymore and the B&P algorithm has to explore many nodes in the B&P search tree in order to decide feasibility. Moreover, the primal heuristic, which is applied at every node, can be very time-consuming here, and it will not always be able to "grab" a feasible solution quickly because the columns of the set-covering formulation (3) contain many jobs and choosing good columns is difficult. The convergence of the column generation is also slow because the pricing oracle requires more time: the search tree is larger, with more branches and more

Table 5 Computational results for Gurobi in function of range

(a) Range $[1, 10]$

m	n	sec	%gap	#no	#opt
5	10	0.008	0.000	0	50
20	25	0.050	0.000	0	50
15	25	54.488	0.364	0	48
10	25	0.866	0.000	0	50
5	25	0.069	0.000	0	50
20	50	0.598	0.000	0	50
15	50	0.580	0.000	0	50
10	50	37.524	0.000	0	50
5	50	0.805	0.000	0	50
20	75	8.132	0.000	0	50
15	75	132.560	0.432	0	44
10	75	319.943	1.167	12	33
5	75	1.920	0.000	0	50
20	100	137.775	0.418	0	44
15	100	266.362	0.546	9	36
10	100	540.919	1.714	21	20
5	100	3.164	0.000	0	50
average		88.574	0.219	42	775

(b) Range $[1, 50]$

m	n	sec	%gap	#no	#opt
5	10	0.009	0.000	0	50
20	25	0.054	0.000	0	50
15	25	0.111	0.000	0	50
10	25	0.424	0.000	0	50
5	25	0.141	0.000	0	50
20	50	221.291	0.318	0	40
15	50	192.571	0.205	0	42
10	50	242.597	0.266	0	40
5	50	2.757	0.000	0	50
20	75	341.912	0.497	0	32
15	75	398.747	0.977	0	29
10	75	460.440	1.932	11	27
5	75	2.623	0.000	0	50
20	100	424.927	0.865	0	29
15	100	467.564	0.765	10	27
10	100	607.572	1.267	22	18
5	100	7.125	0.000	0	50
average		198.286	0.369	43	684

(c) Range $[1, 100]$

m	n	sec	%gap	#no	#opt
5	10	0.009	0.000	0	50
20	25	0.054	0.000	0	50
15	25	0.114	0.000	0	50
10	25	0.328	0.000	0	50
5	25	0.229	0.000	0	50
20	50	405.162	0.430	0	30
15	50	400.285	0.353	0	30
10	50	363.594	0.318	0	34
5	50	3.944	0.000	0	50
20	75	674.408	0.742	0	16
15	75	590.521	1.026	0	20
10	75	587.332	0.883	14	20
5	75	4.118	0.000	0	50
20	100	683.167	1.198	0	15
15	100	574.979	0.921	10	20
10	100	708.439	1.245	22	12
5	100	33.759	0.002	0	49
average		295.908	0.382	46	596

backtracking. As mentioned in Sections 7.3 and 7.4, the highest runtime occurs when the chromatic number is equal to or close to m, i.e., it is not easy to color the graph with m colors. Overall we conclude that our algorithm outperforms Gurobi unless the ratio n/m is very high: our algorithm is better in finding feasible and optimal solutions, and in showing that a given instance is infeasible.

In Table 6 we report the same computational results for Gurobi but now in function of the density of the conflict graph (following the same structure as Table 4). We observe the same phase transitions as in Table 4, but in the most difficult settings Gurobi is not able to identify a feasible solution or even to prove that the instance is infeasible.

7.6. Suggestions for future work

Based on the foregoing, we conclude that it would be interesting to develop a (meta-)heuristic for PMC that is capable of solving or of decreasing the gap for instances with high ratio n/m and where m is close to the chromatic number of the conflict graph. One option might be to extend the scatter search algorithm of Dell'Amico et al. (2008), but it should be noted that it is not always easy to find sufficient feasible solutions in order to run such an algorithm; one way to overcome this issue would be to allow infeasible solutions into the reference set. One can also extend the tabu search algorithm of Alvim and Ribeiro (2004) for PMC; Table 1 shows that this algorithm performs very well on instances that have a high number of jobs on a machine. Initializing this algorithm might be easier since only one feasible solution is needed. To this respect, we mention that the primal heuristic is very good in finding feasible solutions for the instances under consideration, while the MIP solver encounters difficulties.

8. Conclusion

In this paper we have introduced an exact algorithm for parallel machine scheduling with a conflict graph. The algorithm is based on a binary search for the lowest makespan, using a B&P framework and a primal heuristic, and combines methods from bin packing, scheduling and graph coloring (with appropriate modifications), and we use a numerically safe bound (introduced in Held et al. (2012)), which leads to integer-valued profits in the pricing problems (knapsack and KPC), and which avoids floating-point representations of the numbers output by the LP solver, thus avoiding difficulties in assessing the termination condition of the column generation.

The algorithm solves all the benchmark instances of $P||C_{\max}$ with very low average CPU times. For a newly generated dataset with conflict graphs, we have examined the difficulty of the instances as a function of the number of machines and the density of the graph. It turns out that the most difficult instances are those where the number of machines is close to the chromatic number of the graph.

References

Alvim, A.C.F., C.C. Ribeiro. 2004. A hybrid bin-packing heuristic to multiprocessor scheduling. *Experimental and Efficient Algorithms. Proceedings of the Third International Workshop WEA 2004, Angra dos Reis, Brazil, May 25-28*.

Balachandran, B.V., A.A. Zoltners. 1981. An interactive audit-staff scheduling decision support system. *The Accounting Review* **56** 801–812.

Berghman, L., R. Leus, F.C.R. Spieksma. 2014. Optimal solutions for a dock assignment problem with trailer transportation. *Annals of Operations Research* **213** 3–25.

Bodlaender, H.L., K. Jansen, G.J. Woeginger. 1994. Scheduling with incompatible jobs. *Discrete Applied Mathematics* **55** 219–232.

Botha, R.A., J.H.P. Eloff. 2001. Separation of duties for access control enforcement in workflow environments. *IBM Systems Journal* **40** 666–682.

Brélaz, D. 1979. New methods to color the vertices of a graph. *Communications of the ACM* **22** 251–256.

Carraghan, R., P.M. Pardalos. 1990. An exact algorithm for the maximum clique problem. *Operations Research Letters* **9** 375–382.

Dell'Amico, M., M. Iori, S. Martello, M. Monaci. 2008. Heuristic and exact algorithms for the identical parallel machine scheduling problem. *INFORMS Journal on Computing* **20** 333–344.

Dell'Amico, M., S. Martello. 1995. Optimal scheduling of tasks on identical parallel processors. *ORSA Journal on Computing* **7** 191–200.

Elhedhli, S., L. Li, M. Gzara, J. Naoum-Sawaya. 2011. A branch-and-price algorithm for the bin packing problem with conflicts. *INFORMS Journal on Computing* **23** 404–415.

França, P.M., M. Gendreau, G. Laporte, F.M. Müller. 1994. A composite heuristic for the identical parallel machine scheduling problem with minimum makespan objective. *Computers & Operations Research* **21** 205–210.

Frangioni, A., E. Necciari, M.G. Scutella. 2004. A multi-exchange neighborhood for minimum makespan parallel machine scheduling problems. *Journal of Combinatorial Optimization* **8** 195–220.

Gardi, F. 2009. Mutual exclusion scheduling with interval graphs or related classes, part I. *Discrete Applied Mathematics* **157** 19 – 35.

Garey, M.R., D.S. Johnson. 1979. *Computers and Intractability: A Guide to the Theory of NP-completeness*. WH Freeman and Company, New York.

Gaur, D.R., R. Krishnamurti, R. Kohli. 2009. Conflict resolution in the scheduling of television commercials. *Operations Research* **57** 1098–1105.

Giblin, C., S. Hada. 2008. Towards seperation of duties for services. *The 6th International Workshop on SOA & Web Services Best Practices Committee*. OOPSLA, Nashville, October 19.

Gilmore, P.C., R.E. Gomory. 1961. A linear programming approach to the cutting-stock problem. *Operations Research* **9** 849–859.

Graham, R.L. 1966. Bounds for certain multiprocessing anomalies. *Bell System Technical Journal* **45** 1563–1581.

Graham, R.L. 1969. Bounds on multiprocessing timing anomalies. *SIAM Journal on Applied Mathematics* **17** 416–429.

Harvey, W.D., M.L. Ginsberg. 1995. Limited discrepancy search. *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*. Morgan Kaufmann, 607–615.

Hastad, J. 1996. Clique is hard to approximate within $n^{1-\epsilon}$. *Proceedings of the 37th Annual Symposium on Foundations of Computer Science*. IEEE, 627–636.

Held, S., W. Cook, E.C. Sewell. 2012. Maximum-weight stable sets and safe lower bounds for graph coloring. *Mathematical Programming Computation* **4** 363–381.

Hifi, M., N. Otmani. 2012. An algorithm for the disjunctively constrained knapsack problem. *International Journal of Operational Research* **13** 22–43.

Jans, R. 2009. Solving lot-sizing problems on parallel identical machines using symmetry-breaking constraints. *INFORMS Journal on Computing* **21** 123–136.

Johnson, D.S. 1973. Approximation algorithms for combinatorial problems. *Proceedings of the Fifth Annual ACM Symposium on Theory of Computing*. ACM, 38–49.

Johnson, D.S., C.R. Aragon, L.A. McGeoch, C. Schevon. 1991. Optimization by simulated annealing: an experimental evaluation; part II, graph coloring and number partitioning. *Operations Research* **39** 378–406.

Joncour, C., S. Michel, R. Sadykov, D. Sverdlov, F. Vanderbeck. 2010. Column generation based primal heuristics. *Electronic Notes in Discrete Mathematics* **36** 695–702.

Kellerer, H., U. Pferschy, D. Pisinger. 2004. *Knapsack problems*. Springer.

Lawler, E.L., J.K. Lenstra, A.H.G. Rinnooy Kan. 1982. Recent developments in deterministic sequencing and scheduling: A survey. M.A.H. Dempster, J.K. Lenstra, A.H.G. Rinnooy Kan, eds., *Deterministic and Stochastic Scheduling*, *NATO Advanced Study Institutes Series*, vol. 84. Springer Netherlands, 35–73.

Liberti, L. 2012. Symmetry in mathematical programming. J. Lee, S. Leyffer, eds., *Mixed Integer Nonlinear Programming. The IMA Volumes in Mathematics and its Applications 154*. Springer, 263–283.

Manerba, D., R. Mansini. 2015. A branch-and-cut algorithm for the multi-vehicle traveling purchaser problem with pairwise incompatibility constraints. *Networks* **65** 139–154.

Margot, F. 2010. Symmetry in integer linear programming. M. Jünger, T. Liebling, D. Naddef, G. Nemhauser, W. Pulleyblank, G. Reinelt, G. Rinaldi, L. Wolsey, eds., *50 Years of Integer Programming 1958-2008*, chap. 17.

Martello, S., D. Pisinger, P. Toth. 1999. Dynamic programming and strong bounds for the 0-1 knapsack problem. *Management Science* **45** 414–424.

Martello, S., P. Toth. 1977. An upper bound for the zero-one knapsack problem and a branch and bound algorithm. *European Journal of Operational Research* **1** 169–175.

Martin, R.K. 1999. *Large Scale Linear and Integer Optimization: A Unified Approach*. Springer.

Mehrotra, A., M. Trick. 1996. A column generation approach for graph coloring. *INFORMS Journal on Computing* **8** 344–354.

Muritiba, A.E.F., M. Iori, E. Malaguti, P. Toth. 2010. Algorithms for the bin packing problem with conflicts. *INFORMS Journal on Computing* **22** 401–415.

Östergård, P.R.J. 2001. A new algorithm for the maximum-weight clique problem. *Nordic Journal of Computing* **8** 424–436.

Pferschy, U., J. Schauer. 2009. The knapsack problem with conflict graphs. *Journal of Graph Algorithms and Applications* **13** 233–249.

ROADEF. 2011. Google ROADEF/EURO challenge 2012: Machine reassignment. Available online at `http://challenge.roadef.org/2012/files/problem_definition_v1.pdf`.

Sadykov, R., F. Vanderbeck. 2013. Bin packing with conflicts: a generic branch-and-price algorithm. *INFORMS Journal on Computing* **25** 244–255.

Sherali, H.D., J.C. Smith. 2001. Improving discrete model representations via symmetry considerations. *Management Science* **47** 1396–1407.

Sun, H.Y., W.L. Zhao, J. Yang. 2010. Managing conflict of interest in service composition. *Lecture Notes in Computer Science* **6426** 273–290.

Vanderbeck, F. 2000. On Dantzig-Wolfe decomposition in integer programming and ways to perform branching in a branch-and-price algorithm. *Operations Research* **48** 111–128.

Zykov, A.A. 1949. On some properties of linear complexes. *Matematicheskii Sbornik* **66** 163–188.

Appendix. Symmetry breaking

With symmetry breaking constraints (SBCs) we try to exclude isomorphic solutions in the exploration of the solution space scanned by a MIP solver. Symmetry is an important problem for such solvers, because many subproblems encountered in the B&B tree can be equivalent and this results in duplication of computational effort. For a survey paper on symmetry in integer programming we refer to Margot (2010) and Liberti (2012). For some applications of SBCs to scheduling and other problems we mention Berghman et al. (2014), Jans (2009) and Sherali and Smith (2001).

The intuitive formulation (1) for PMC assigns jobs to identical machines and thus many alternative (sub-)optimal solutions can be constructed simply by interchanging machines. In this appendix we present several SBCs from the literature that can be applied to formulation (1) (for PMC, and thus also for $P||C_{\max}$). For all these SBCs we provide extensive computational results.

The first class of SBCs that we consider enforces that the number of jobs that is scheduled on machine i is at least as high as the number of jobs on machine $i+1$. This can be stated as follows:

$$\sum_{j\in J} x_{ij} \geq \sum_{j\in J} x_{i+1,j} \qquad \forall i \in M \setminus \{m\}. \tag{14}$$

The next class of SBCs is based on Jans (2009). For each set S of jobs scheduled on a machine a unique number $\sum_{j\in S} 2^j$ is assigned. The machines are then ordered by decreasing value of this number. This leads to the following set of constraints:

$$\sum_{j\in J} 2^j x_{ij} \geq \sum_{j\in J} 2^j_{i+1,j} \qquad \forall i \in M \setminus \{m\}. \tag{15}$$

The third type of symmetry breaking that we have considered, is based on the reasoning that every job is scheduled on the machine with lowest index, giving priority to the lowest-indexed jobs. This means that a job j can only be scheduled on a specific machine i if at least one of the jobs of set $\{1,\ldots,j-1\}$ is planned on machine $i-1$. We formulate this as follows:

$$x_{ij} \leq \sum_{k=1}^{j-1} x_{i-1,k} \qquad i \in M \setminus \{1\}, \forall j \in J \setminus \{1\} \tag{16}$$

The last class of SBCs that we have examined, enforces that the first m jobs have to be scheduled on a specific subset of machines. Concretely, we demand that job j, with $j \leq m$, be assigned to a machine in the subset $\{1,\ldots,j\}$:

$$\sum_{i=1}^{j} x_{ij} = 1 \qquad \forall j \in \{1,\ldots,m\}. \tag{17}$$

We refer to these constraint sets as SBC1, SBC2, SBC3 and SBC4, respectively. It can be seen that SBC2 and SBC3 eliminate all symmetry in the formulation, whereas SBC1 and SBC4 might still allow multiple equivalent solutions. We have tested these SBCs on the instances with conflict graphs. For each range of processing times, number of machines and number of jobs we provide in Tables 7–10 the average CPU time (sec) in seconds, the average percentage gap (%gap) and the number of optimal solutions (#opt). Tables 11–14 contain the same information but now in function of the density of the conflict graph. These results can be

compared with Tables 3 and 4, respectively (which show the results without symmetry breaking). Overall, none of the constraint sets has a major impact on the performance of the formulation; SBC2 and SBC4 seem to be best, with minor improvements compared to the formulations without SBCs in the duration range $[1, 100]$ and for low graph densities, but the differences are not really significant.

Table 6 Computational results for Gurobi in function of density

(a) $d = 0.1$

m	n	sec	%gap	#no	#opt
5	10	0.007	0.000	0	30
20	25	0.036	0.000	0	30
15	25	30.197	0.303	0	29
10	25	1.535	0.000	0	30
5	25	0.061	0.000	0	30
20	50	221.447	0.281	0	23
15	50	13.998	0.000	0	30
10	50	4.020	0.000	0	30
5	50	0.263	0.000	0	30
20	75	135.717	0.083	0	26
15	75	33.266	0.013	0	29
10	75	1.669	0.000	0	30
5	75	1.451	0.000	0	30
20	100	34.656	0.013	0	29
15	100	2.665	0.000	0	30
10	100	1.721	0.000	0	30
5	100	61.618	0.003	0	29
average		32.019	0.041	0	495

(b) $d = 0.2$

m	n	sec	%gap	#no	#opt
5	10	0.007	0.000	0	30
20	25	0.048	0.000	0	30
15	25	30.081	0.303	0	29
10	25	0.166	0.000	0	30
5	25	0.155	0.000	0	30
20	50	227.181	0.255	0	23
15	50	16.275	0.000	0	30
10	50	12.635	0.000	0	30
5	50	10.886	0.000	0	30
20	75	178.438	0.082	0	26
15	75	143.335	0.049	0	26
10	75	121.505	0.026	0	27
5	75	11.256	0.000	0	30
20	100	260.230	0.103	0	23
15	100	63.962	0.011	0	29
10	100	393.254	0.097	0	20
5	100	11.095	0.000	0	30
average		87.089	0.054	0	473

(c) $d = 0.3$

m	n	sec	%gap	#no	#opt
5	10	0.009	0.000	0	30
20	25	0.054	0.000	0	30
15	25	2.830	0.000	0	30
10	25	0.541	0.000	0	30
5	25	0.310	0.000	0	30
20	50	217.918	0.235	0	24
15	50	92.932	0.055	0	28
10	50	124.639	0.040	0	27
5	50	1.141	0.000	0	30
20	75	308.212	0.186	0	21
15	75	273.129	0.104	0	24
10	75	550.883	0.388	0	13
5	75	1.503	0.000	0	30
20	100	374.377	0.190	0	21
15	100	472.808	0.250	0	18
10	100	899.907	4.685	5	0
5	100	0.236	0.000	0	30
average		195.378	0.318	5	416

(d) $d = 0.4$

m	n	sec	%gap	#no	#opt
5	10	0.010	0.000	0	30
20	25	0.061	0.000	0	30
15	25	25.745	0.000	0	30
10	25	0.191	0.000	0	30
5	25	0.137	0.000	0	30
20	50	186.126	0.196	0	25
15	50	356.777	0.286	0	19
10	50	559.160	0.484	0	12
5	50	0.180	0.000	0	30
20	75	472.976	0.585	0	15
15	75	603.134	0.760	0	10
10	75	899.972	10.699	17	0
5	75	0.119	0.000	0	30
20	100	589.059	0.856	0	11
15	100	742.077	2.328	0	6
10	100	900.000	—	30	0
5	100	0.240	0.000	0	30
average		313.880	0.656	47	338

(e) $d = 0.5$

m	n	sec	%gap	#no	#opt
5	10	0.011	0.000	0	30
20	25	0.065	0.000	0	30
15	25	2.335	0.000	0	30
10	25	0.263	0.000	0	30
5	25	0.069	0.000	0	30
20	50	192.412	0.278	0	25
15	50	509.078	0.589	0	15
10	50	372.403	0.449	0	25
5	50	0.041	0.000	0	30
20	75	612.077	1.128	0	10
15	75	816.849	3.132	0	4
10	75	705.496	0.000	20	10
5	75	0.105	0.000	0	30
20	100	818.127	2.973	0	4
15	100	899.996	12.195	29	0
10	100	900.000	—	30	0
5	100	0.223	0.000	0	30
average		342.915	0.623	79	333

Table 7 Computational results for SBC1 in function of range

(a) Range $[1, 10]$

m	n	sec	%gap	#no	#opt
5	10	0.019	0.000	0	50
20	25	0.192	0.000	0	50
15	25	87.264	0.682	0	47
10	25	0.215	0.000	0	50
5	25	80.035	0.493	0	46
20	50	76.935	0.535	0	46
15	50	7.972	0.000	0	50
10	50	195.306	1.590	0	40
5	50	13.013	0.000	0	50
20	75	46.145	0.095	0	49
15	75	191.588	0.759	0	40
10	75	426.738	0.229	20	27
5	75	107.257	0.000	5	45
20	100	241.699	0.894	0	39
15	100	371.169	0.000	10	31
10	100	544.833	1.631	25	20
5	100	50.311	0.000	1	49
average		143.570	0.380	61	729

(b) Range $[1, 50]$

m	n	sec	%gap	#no	#opt
5	10	0.026	0.000	0	50
20	25	0.210	0.000	0	50
15	25	3.668	0.000	0	50
10	25	3.435	0.000	0	50
5	25	294.583	0.597	0	37
20	50	719.770	1.815	0	11
15	50	426.279	0.861	0	29
10	50	412.864	2.056	0	28
5	50	77.586	0.009	0	49
20	75	630.704	1.434	0	18
15	75	586.455	1.645	0	19
10	75	605.727	0.479	20	18
5	75	95.370	0.000	4	46
20	100	668.875	1.992	0	16
15	100	642.352	1.677	10	17
10	100	698.277	1.453	25	12
5	100	94.056	0.004	3	46
average		350.602	0.810	62	546

(c) Range $[1, 100]$

m	n	sec	%gap	#no	#opt
5	10	0.028	0.000	0	50
20	25	0.225	0.000	0	50
15	25	43.628	0.040	0	48
10	25	3.632	0.000	0	50
5	25	614.044	1.125	0	23
20	50	760.023	1.895	0	9
15	50	721.581	1.288	0	15
10	50	681.766	2.217	0	15
5	50	101.846	0.027	0	46
20	75	855.913	1.895	0	4
15	75	796.704	1.720	0	8
10	75	702.574	0.531	20	13
5	75	125.243	0.000	6	44
20	100	804.542	2.286	0	8
15	100	785.366	1.795	10	8
10	100	735.425	1.265	27	10
5	100	100.239	0.004	2	46
average		460.752	0.945	65	447

Table 8 Computational results for SBC2 in function of range

(a) Range $[1, 10]$

m	n	sec	%gap	#no	#opt
5	10	0.004	0.000	0	50
20	25	0.036	0.000	0	50
15	25	0.035	0.000	0	50
10	25	0.054	0.000	0	50
5	25	0.054	0.000	0	50
20	50	0.469	0.000	0	50
15	50	0.920	0.000	0	50
10	50	137.547	0.456	0	45
5	50	1.314	0.000	0	50
20	75	8.481	0.000	0	50
15	75	175.127	0.565	0	42
10	75	367.521	1.838	17	30
5	75	3.701	0.000	0	50
20	100	162.272	0.482	0	43
15	100	310.296	0.316	10	36
10	100	542.259	2.170	21	21
5	100	3.593	0.000	0	50
average		100.805	0.264	48	767

(b) Range $[1, 50]$

m	n	sec	%gap	#no	#opt
5	10	0.004	0.000	0	50
20	25	0.039	0.000	0	50
15	25	0.037	0.000	0	50
10	25	0.074	0.000	0	50
5	25	0.122	0.000	0	50
20	50	39.424	0.029	0	49
15	50	150.686	0.158	0	43
10	50	358.964	1.274	0	31
5	50	4.664	0.000	0	50
20	75	357.698	0.441	0	32
15	75	410.330	0.862	0	28
10	75	526.709	1.547	17	21
5	75	3.282	0.000	0	50
20	100	451.491	1.080	0	28
15	100	472.008	0.781	10	26
10	100	601.215	1.739	23	17
5	100	9.958	0.000	0	50
average		199.218	0.402	50	675

(c) Range $[1, 100]$

m	n	sec	%gap	#no	#opt
5	10	0.005	0.000	0	50
20	25	0.040	0.000	0	50
15	25	0.041	0.000	0	50
10	25	0.111	0.000	0	50
5	25	0.245	0.000	0	50
20	50	41.952	0.000	0	49
15	50	402.627	0.003	0	32
10	50	424.402	0.012	0	30
5	50	5.458	0.000	0	50
20	75	708.680	0.007	0	14
15	75	544.797	0.009	0	24
10	75	595.998	0.002	20	18
5	75	5.874	0.000	0	50
20	100	649.550	0.010	0	17
15	100	561.225	0.010	10	20
10	100	685.913	0.017	23	13
5	100	30.214	0.000	0	49
average		273.949	0.374	53	616

Table 9 Computational results for SBC3 in function of range

(a) Range $[1, 10]$

m	n	sec	%gap	#no	#opt
5	10	0.016	0.000	0	50
20	25	0.688	0.000	0	50
15	25	7.760	0.000	0	50
10	25	19.050	0.125	0	49
5	25	0.164	0.000	0	50
20	50	21.830	0.000	0	50
15	50	4.052	0.000	0	50
10	50	182.084	1.197	0	40
5	50	8.353	0.000	0	50
20	75	24.674	0.000	0	50
15	75	164.866	0.488	0	43
10	75	382.845	0.390	19	29
5	75	34.174	0.000	1	49
20	100	178.777	0.489	0	43
15	100	277.558	0.191	10	37
10	100	543.976	1.453	24	20
5	100	25.491	0.000	0	50
average		110.374	0.217	54	760

(b) Range $[1, 50]$

m	n	sec	%gap	#no	#opt
5	10	0.019	0.000	0	50
20	25	0.743	0.000	0	50
15	25	5.587	0.000	0	50
10	25	57.080	0.049	0	48
5	25	0.935	0.000	0	50
20	50	551.813	1.005	0	20
15	50	264.712	0.383	0	36
10	50	360.740	1.583	0	31
5	50	18.010	0.000	0	50
20	75	396.190	0.516	0	30
15	75	452.256	0.971	0	26
10	75	541.224	0.580	19	21
5	75	30.043	0.000	0	50
20	100	503.697	1.014	0	24
15	100	507.660	0.786	10	24
10	100	629.576	1.440	25	16
5	100	33.945	0.000	0	50
average		256.131	0.454	54	626

(c) Range $[1, 100]$

m	n	sec	%gap	#no	#opt
5	10	0.021	0.000	0	50
20	25	0.874	0.000	0	50
15	25	6.582	0.000	0	50
10	25	116.837	0.106	0	46
5	25	1.488	0.000	0	50
20	50	705.801	1.333	0	12
15	50	517.094	0.498	0	24
10	50	452.998	1.435	0	27
5	50	52.863	0.008	0	49
20	75	767.133	0.826	0	9
15	75	673.068	1.131	0	15
10	75	648.597	1.996	16	15
5	75	38.961	0.000	0	50
20	100	658.827	1.125	0	17
15	100	627.882	1.022	10	18
10	100	713.029	2.950	23	11
5	100	77.283	0.002	0	49
average		356.432	0.639	49	542

Table 10 Computational results for SBC4 in function of range

(a) Range $[1, 10]$

m	n	sec	%gap	#no	#opt
5	10	0.003	0.000	0	50
20	25	0.015	0.000	0	50
15	25	0.023	0.000	0	50
10	25	0.101	0.000	0	50
5	25	0.047	0.000	0	50
20	50	0.347	0.000	0	50
15	50	0.513	0.000	0	50
10	50	180.700	1.038	0	40
5	50	1.482	0.000	0	50
20	75	2.601	0.000	0	50
15	75	85.157	0.288	0	46
10	75	369.570	1.358	15	30
5	75	4.626	0.000	0	50
20	100	135.204	0.489	0	43
15	100	247.495	0.191	10	37
10	100	540.881	1.712	21	21
5	100	3.773	0.000	0	50
average		92.502	0.243	46	767

(b) Range $[1, 50]$

m	n	sec	%gap	#no	#opt
5	10	0.003	0.000	0	50
20	25	0.016	0.000	0	50
15	25	0.026	0.000	0	50
10	25	0.075	0.000	0	50
5	25	0.150	0.000	0	50
20	50	66.098	0.058	0	48
15	50	146.371	0.198	0	43
10	50	365.787	1.402	0	31
5	50	7.132	0.000	0	50
20	75	299.306	0.439	0	35
15	75	416.351	0.805	0	29
10	75	535.101	1.539	15	22
5	75	3.648	0.000	0	50
20	100	430.767	0.893	0	28
15	100	470.334	0.567	10	27
10	100	577.723	1.174	22	19
5	100	9.354	0.000	0	50
average		195.779	0.373	47	682

(c) Range $[1, 100]$

m	n	sec	%gap	#no	#opt
5	10	0.003	0.000	0	50
20	25	0.017	0.000	0	50
15	25	0.028	0.000	0	50
10	25	0.129	0.000	0	50
5	25	0.452	0.000	0	50
20	50	72.995	0.014	0	49
15	50	460.466	0.430	0	26
10	50	422.857	1.465	0	28
5	50	11.434	0.000	0	50
20	75	616.180	0.620	0	19
15	75	515.174	0.998	0	25
10	75	612.086	1.074	17	17
5	75	9.223	0.000	0	50
20	100	625.385	1.076	0	18
15	100	547.835	0.767	10	21
10	100	697.570	1.805	21	12
5	100	17.410	0.000	0	50
average		271.132	0.435	48	615

Table 11 Computational results for SBC1 in function of density

(a) $d = 0.1$

m	n	sec	%gap	#no	#opt
5	10	0.019	0.000	0	30
20	25	0.113	0.000	0	30
15	25	73.173	0.336	0	28
10	25	0.163	0.000	0	30
5	25	401.560	1.147	0	17
20	50	430.962	0.779	0	17
15	50	149.652	0.075	0	27
10	50	10.098	0.000	0	30
5	50	0.584	0.000	0	30
20	75	294.453	0.188	0	23
15	75	133.767	0.027	0	28
10	75	23.740	0.000	0	30
5	75	8.638	0.000	0	30
20	100	163.120	0.026	0	28
15	100	114.855	0.020	0	28
10	100	28.844	0.000	0	30
5	100	141.790	0.013	0	27
average		116.208	0.154	0	463

(b) $d = 0.2$

m	n	sec	%gap	#no	#opt
5	10	0.023	0.000	0	30
20	25	0.174	0.000	0	30
15	25	13.053	0.000	0	30
10	25	2.789	0.000	0	30
5	25	165.309	0.158	0	27
20	50	399.285	1.018	0	17
15	50	254.554	0.226	0	24
10	50	238.010	0.064	0	25
5	50	311.803	0.060	0	25
20	75	414.507	0.488	0	17
15	75	373.425	0.191	0	19
10	75	357.777	0.134	0	21
5	75	535.088	0.000	15	15
20	100	534.905	0.512	0	13
15	100	469.983	0.287	0	17
10	100	568.764	0.278	0	12
5	100	265.144	0.000	6	24
average		288.506	0.210	21	376

(c) $d = 0.3$

m	n	sec	%gap	#no	#opt
5	10	0.025	0.000	0	30
20	25	0.231	0.000	0	30
15	25	41.897	0.278	0	29
10	25	8.386	0.000	0	30
5	25	342.464	0.511	0	22
20	50	574.253	1.414	0	11
15	50	359.977	0.341	0	21
10	50	377.486	0.230	0	18
5	50	8.104	0.000	0	30
20	75	575.105	0.786	0	12
15	75	604.074	0.686	0	10
10	75	710.214	1.106	0	7
5	75	2.467	0.000	0	30
20	100	574.795	0.830	0	13
15	100	639.208	0.753	0	10
10	100	899.950	7.526	17	0
5	100	0.243	0.000	0	30
average		336.405	0.620	17	333

(d) $d = 0.4$

m	n	sec	%gap	#no	#opt
5	10	0.028	0.000	0	30
20	25	0.230	0.000	0	30
15	25	32.686	0.034	0	29
10	25	0.684	0.000	0	30
5	25	280.503	0.547	0	22
20	50	554.801	1.641	0	12
15	50	552.063	1.122	0	12
10	50	624.315	1.075	0	10
5	50	0.200	0.000	0	30
20	75	617.598	1.535	0	10
15	75	613.438	1.284	0	10
10	75	900.000	—	30	0
5	75	0.134	0.000	0	30
20	100	685.853	2.036	0	9
15	100	874.101	4.502	0	1
10	100	900.000	—	30	0
5	100	0.271	0.000	0	30
average		390.406	0.918	60	295

(e) $d = 0.5$

m	n	sec	%gap	#no	#opt
5	10	0.028	0.000	0	30
20	25	0.297	0.000	0	30
15	25	63.457	0.556	0	29
10	25	0.114	0.000	0	30
5	25	457.934	1.328	0	18
20	50	635.244	2.225	0	9
15	50	610.139	1.818	0	10
10	50	899.984	8.402	0	0
5	50	0.049	0.000	0	30
20	75	652.940	2.711	0	9
15	75	899.874	4.687	0	0
10	75	900.000	—	30	0
5	75	0.125	0.000	0	30
20	100	899.852	5.214	0	0
15	100	900.000	0.000	30	0
10	100	900.000	—	30	0
5	100	0.229	0.000	0	30
average		460.016	1.924	90	255

Table 12 Computational results for SBC2 in function of density

(a) $d = 0.1$

m	n	sec	%gap	#no	#opt
5	10	0.004	0.000	0	30
20	25	0.025	0.000	0	30
15	25	0.025	0.000	0	30
10	25	0.070	0.000	0	30
5	25	0.068	0.000	0	30
20	50	84.948	0.048	0	29
15	50	15.324	0.000	0	30
10	50	0.831	0.000	0	30
5	50	0.310	0.000	0	30
20	75	133.475	0.069	0	26
15	75	3.755	0.000	0	30
10	75	4.451	0.000	0	30
5	75	1.732	0.000	0	30
20	100	31.918	0.000	0	30
15	100	4.611	0.000	0	30
10	100	1.847	0.000	0	30
5	100	56.113	0.003	0	29
average		19.971	0.007	0	504

(b) $d = 0.2$

m	n	sec	%gap	#no	#opt
5	10	0.004	0.000	0	30
20	25	0.030	0.000	0	30
15	25	0.030	0.000	0	30
10	25	0.061	0.000	0	30
5	25	0.208	0.000	0	30
20	50	5.835	0.000	0	30
15	50	25.798	0.000	0	30
10	50	17.019	0.000	0	30
5	50	17.999	0.000	0	30
20	75	185.973	0.067	0	26
15	75	134.262	0.037	0	27
10	75	95.343	0.016	0	28
5	75	19.362	0.000	0	30
20	100	233.378	0.065	0	25
15	100	46.658	0.000	0	30
10	100	369.270	0.122	0	19
5	100	16.368	0.000	0	30
average		68.682	0.018	0	485

(c) $d = 0.3$

m	n	sec	%gap	#no	#opt
5	10	0.004	0.000	0	30
20	25	0.037	0.000	0	30
15	25	0.037	0.000	0	30
10	25	0.078	0.000	0	30
5	25	0.319	0.000	0	30
20	50	34.710	0.023	0	29
15	50	111.314	0.056	0	27
10	50	103.215	0.013	0	29
5	50	0.688	0.000	0	30
20	75	331.811	0.190	0	21
15	75	277.968	0.113	0	24
10	75	583.932	0.447	0	11
5	75	0.173	0.000	0	30
20	100	401.317	0.217	0	19
15	100	480.400	0.299	0	16
10	100	899.927	7.221	9	0
5	100	0.190	0.000	0	30
average		189.772	0.384	9	416

(d) $d = 0.4$

m	n	sec	%gap	#no	#opt
5	10	0.004	0.000	0	30
20	25	0.046	0.000	0	30
15	25	0.044	0.000	0	30
10	25	0.079	0.000	0	30
5	25	0.098	0.000	0	30
20	50	4.386	0.000	0	30
15	50	288.333	0.191	0	22
10	50	593.802	0.533	0	11
5	50	0.038	0.000	0	30
20	75	533.253	0.535	0	13
15	75	582.908	0.651	0	11
10	75	899.988	17.540	24	0
5	75	0.094	0.000	0	30
20	100	605.392	0.805	0	11
15	100	807.546	2.439	0	6
10	100	900.000	—	30	0
5	100	0.156	0.000	0	30
average		306.833	0.570	54	344

(e) $d = 0.5$

m	n	sec	%gap	#no	#opt
5	10	0.004	0.000	0	30
20	25	0.053	0.000	0	30
15	25	0.052	0.000	0	30
10	25	0.110	0.000	0	30
5	25	0.009	0.000	0	30
20	50	6.527	0.000	0	30
15	50	482.951	0.526	0	16
10	50	819.988	4.327	0	6
5	50	0.024	0.000	0	30
20	75	606.919	0.993	0	10
15	75	884.865	3.155	0	2
10	75	900.000	—	30	0
5	75	0.067	0.000	0	30
20	100	833.516	3.184	0	3
15	100	900.000	—	30	0
10	100	877.934	0.000	28	2
5	100	0.115	0.000	0	30
average		371.361	0.866	88	309

Table 13 Computational results for SBC3 in function of density

(a) $d = 0.1$

m	n	sec	%gap	#no	#opt
5	10	0.015	0.000	0	30
20	25	0.618	0.000	0	30
15	25	2.511	0.000	0	30
10	25	88.919	0.255	0	28
5	25	0.086	0.000	0	30
20	50	315.274	0.505	0	20
15	50	29.372	0.000	0	30
10	50	2.138	0.000	0	30
5	50	0.628	0.000	0	30
20	75	205.724	0.149	0	24
15	75	40.720	0.013	0	29
10	75	16.524	0.000	0	30
5	75	5.680	0.000	0	30
20	100	25.632	0.000	0	30
15	100	11.609	0.000	0	30
10	100	5.510	0.000	0	30
5	100	119.301	0.003	0	29
average		51.192	0.054	0	490

(b) $d = 0.2$

m	n	sec	%gap	#no	#opt
5	10	0.016	0.000	0	30
20	25	0.663	0.000	0	30
15	25	1.902	0.000	0	30
10	25	44.761	0.046	0	29
5	25	1.228	0.000	0	30
20	50	365.821	0.599	0	19
15	50	86.290	0.019	0	29
10	50	20.497	0.000	0	30
5	50	126.253	0.014	0	29
20	75	249.016	0.150	0	23
15	75	193.195	0.074	0	25
10	75	176.639	0.042	0	25
5	75	162.401	0.000	1	29
20	100	282.746	0.115	0	24
15	100	157.943	0.020	0	28
10	100	438.844	0.111	0	17
5	100	107.502	0.000	0	30
average		142.101	0.070	1	457

(c) $d = 0.3$

m	n	sec	%gap	#no	#opt
5	10	0.019	0.000	0	30
20	25	0.752	0.000	0	30
15	25	7.583	0.000	0	30
10	25	93.494	0.109	0	28
5	25	2.427	0.000	0	30
20	50	424.382	0.747	0	16
15	50	164.478	0.097	0	25
10	50	152.156	0.037	0	27
5	50	4.812	0.000	0	30
20	75	360.833	0.256	0	19
15	75	448.363	0.256	0	17
10	75	627.963	0.528	0	10
5	75	3.462	0.000	0	30
20	100	450.098	0.333	0	17
15	100	541.828	0.338	0	14
10	100	899.947	8.339	12	0
5	100	0.322	0.000	0	30
average		246.054	0.464	12	383

(d) $d = 0.4$

m	n	sec	%gap	#no	#opt
5	10	0.022	0.000	0	30
20	25	0.900	0.000	0	30
15	25	5.989	0.000	0	30
10	25	47.104	0.039	0	29
5	25	0.449	0.000	0	30
20	50	510.470	0.968	0	14
15	50	426.329	0.442	0	16
10	50	584.950	0.599	0	11
5	50	0.264	0.000	0	30
20	75	544.307	0.514	0	13
15	75	610.628	0.763	0	10
10	75	899.985	13.479	24	0
5	75	0.212	0.000	0	30
20	100	641.274	0.920	0	10
15	100	743.788	2.307	0	7
10	100	900.000	—	30	0
5	100	0.361	0.000	0	30
average		348.061	0.608	54	320

(e) $d = 0.5$

m	n	sec	%gap	#no	#opt
5	10	0.020	0.000	0	30
20	25	0.908	0.000	0	30
15	25	15.227	0.000	0	30
10	25	47.334	0.019	0	29
5	25	0.122	0.000	0	30
20	50	516.460	1.079	0	13
15	50	603.294	0.910	0	10
10	50	899.964	6.389	0	0
5	50	0.085	0.000	0	30
20	75	620.115	1.167	0	10
15	75	857.410	3.212	0	3
10	75	900.000	—	30	0
5	75	0.209	0.000	0	30
20	100	835.752	3.012	0	3
15	100	900.000	—	30	0
10	100	900.000	0.000	30	0
5	100	0.378	0.000	0	30
average		417.487	1.128	90	278

Table 14 Computational results for SBC4 in function of density

(a) $d = 0.1$

m	n	sec	%gap	#no	#opt
5	10	0.003	0.000	0	30
20	25	0.009	0.000	0	30
15	25	0.019	0.000	0	30
10	25	0.185	0.000	0	30
5	25	0.050	0.000	0	30
20	50	102.412	0.048	0	29
15	50	37.570	0.019	0	29
10	50	0.949	0.000	0	30
5	50	0.276	0.000	0	30
20	75	66.329	0.016	0	29
15	75	34.133	0.013	0	29
10	75	1.917	0.000	0	30
5	75	1.424	0.000	0	30
20	100	35.621	0.013	0	29
15	100	3.711	0.000	0	30
10	100	1.258	0.000	0	30
5	100	34.458	0.000	0	30
average		18.843	0.006	0	505

(b) $d = 0.2$

m	n	sec	%gap	#no	#opt
5	10	0.003	0.000	0	30
20	25	0.012	0.000	0	30
15	25	0.021	0.000	0	30
10	25	0.059	0.000	0	30
5	25	0.228	0.000	0	30
20	50	63.224	0.048	0	29
15	50	23.458	0.000	0	30
10	50	9.231	0.000	0	30
5	50	31.740	0.000	0	30
20	75	175.621	0.114	0	25
15	75	51.352	0.000	0	30
10	75	124.044	0.024	0	27
5	75	27.326	0.000	0	30
20	100	160.573	0.039	0	27
15	100	40.482	0.000	0	30
10	100	366.456	0.098	0	20
5	100	15.851	0.000	0	30
average		64.099	0.019	0	488

(c) $d = 0.3$

m	n	sec	%gap	#no	#opt
5	10	0.003	0.000	0	30
20	25	0.016	0.000	0	30
15	25	0.023	0.000	0	30
10	25	0.068	0.000	0	30
5	25	0.691	0.000	0	30
20	50	53.540	0.023	0	29
15	50	108.569	0.060	0	27
10	50	104.545	0.026	0	28
5	50	1.318	0.000	0	30
20	75	238.886	0.118	0	24
15	75	268.166	0.088	0	25
10	75	601.994	0.344	0	12
5	75	0.226	0.000	0	30
20	100	387.741	0.297	0	19
15	100	455.352	0.221	0	18
10	100	899.917	5.497	6	0
5	100	0.236	0.000	0	30
average		183.605	0.332	6	422

(d) $d = 0.4$

m	n	sec	%gap	#no	#opt
5	10	0.003	0.000	0	30
20	25	0.020	0.000	0	30
15	25	0.030	0.000	0	30
10	25	0.077	0.000	0	30
5	25	0.104	0.000	0	30
20	50	5.471	0.000	0	30
15	50	364.080	0.306	0	18
10	50	600.817	0.534	0	11
5	50	0.048	0.000	0	30
20	75	446.872	0.499	0	16
15	75	601.829	0.628	0	10
10	75	899.976	9.674	17	0
5	75	0.100	0.000	0	30
20	100	585.295	0.820	0	11
15	100	709.894	1.813	0	7
10	100	900.000	—	30	0
5	100	0.202	0.000	0	30
average		300.872	0.570	47	343

(e) $d = 0.5$

m	n	sec	%gap	#no	#opt
5	10	0.003	0.000	0	30
20	25	0.022	0.000	0	30
15	25	0.034	0.000	0	30
10	25	0.119	0.000	0	30
5	25	0.010	0.000	0	30
20	50	7.752	0.000	0	30
15	50	478.573	0.661	0	15
10	50	900.031	5.948	0	0
5	50	0.030	0.000	0	30
20	75	602.437	1.017	0	10
15	75	738.990	2.756	0	6
10	75	900.000	—	30	0
5	75	0.085	0.000	0	30
20	100	816.365	2.928	0	3
15	100	900.000	—	30	0
10	100	859.326	0.000	28	2
5	100	0.148	0.000	0	30
average		364.937	0.946	88	306

FACULTY OF ECONOMICS AND BUSINESS
Naamsestraat 69 bus 3500
3000 LEUVEN, BELGIË
tel. + 32 16 32 66 12
fax + 32 16 32 67 91
info@econ.kuleuven.be
www.econ.kuleuven.be