Solving a Multiprocessor Problem by Column Generation and Branch-and-price

António J. S. T. Duarte*, J. M. Valério de Carvalho**

**Polytechnic Institute of Bragança, Bragança, Portugal (e-mail: aduarte@ipb.pt)*
***University of Minho, Braga, Portugal (e-mail: vc@dps.uminho.pt)*

Abstract: This work presents an algorithm for solving exactly a scheduling problem with identical parallel machines and malleable tasks, subject to arbitrary release dates and due dates. The objective is to minimize a function of late work and setup costs. A task is malleable if we can freely change the set of machines assigned to its processing over the time horizon. We present an integer programming model, a Dantzig-Wolfe decomposition reformulation and its solution by column generation. We also developed an equivalent network flow model, used for the branching phase. Finally, we carried out extensive computational tests to verify the algorithm's efficiency and to determine the model's sensitivity to instance size parameters: the number of machines, the number of tasks and the size of the planning horizon.

1. PROBLEM DEFINITION

Given a set of tasks (T_1, T_2, …, T_n) and a set of machines (P_1, P_2, …, P_m), the objective is to find a feasible schedule that minimizes a function of the number of setups and the amount of late work. The tasks allow for preemptions, multiprocessing and are malleable (Blazewicz *et al*., 2004). This last property means that we can freely change the set of machines processing a given task over the time horizon, as shown in the next machine schedule:

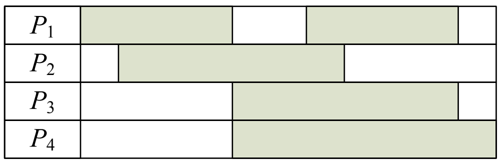

Fig. 1. Scheduling of a malleable task

The task (in shadow) is scheduled in four machines, and the number of machines assigned to the task may vary arbitrarily over time.

1.1 Machines

The problem considers a set of m identical machines, $P = \{P_1, P_2, \ldots, P_m\}$. Each machine can only process one task at a time and they can be preempted at arbitrary moments. Because of computational complexity issues, in our models we will only allow preemptions at integer time units. This can be done without loss of generality, because the time unit can be freely chosen, according to the problem requirements and the available computational resources.

1.2 Tasks

For the definition of the task environment, we use a set of n tasks, $T = \{T_1, T_2, \ldots, T_n\}$. Each task has a processing time (p_j), a release date (r_j), a due date (d_j) and a weight for the calculation of late work cost (w_j).

Although the values of p_j, r_j and d_j can be any nonnegative numbers, for consistency with the machine environment, we limit these values to nonnegative integers.

A machine schedule can be seen as a sequence of unit fragments of different tasks. If two consecutive fragments belong to the same task there is no setup cost, otherwise, a setup cost is incurred. Although a setup requires, among other resources, a certain amount of time, in our models, we assume that setup times are very short when compared to production runs. Therefore, we will only minimize the total number of setups.

1.3 Objective Function

Let c_k be the cost of schedule k (a single machine schedule). We define c_k in the following way:

$$c_k = l_k + s_k \tag{1}$$

In the previous equation l_k is the cost incurred because of late work in schedule k and s_k is the number of setups in the schedule. The value of l_k is calculated according to the late work in the schedule and to the values of the w_j constants. The values for w_j must be chosen wisely because there is an implicit trade-off with the setup costs. Also, the model for calculation of l_k can be subject of discussion.

For the calculation of l_k we choose a function that linearly increases the penalty as the task completion time is more distant from the task due date. The total late work penalty for a given machine schedule can be computed as follows:

$$l_k = \sum_{j=1}^{n} \sum_{l \in L_{jk}} w_j \left(C_l - d_j \right) \qquad (2)$$

In the above equation, L_{jk} represents the set of late fragments belonging to task T_j present at schedule *k* and C_l is the completion time for each one of those fragments. This function can be replaced with any function compatible with the subproblem algorithm, which will be presented later on this paper.

Note that the total cost of a given schedule is the sum of costs of every machine and that it is possible to calculate the contribution of each machine schedule to the total cost without knowing the schedules of the other machines.

2. PROBLEM FORMULATION

2.1 A Simple Network Flow Model

We will show below that the problem presented in the previous section can be modeled as a network flow problem and solved with a suitable algorithm. Although the network flow model always provides valid solutions for the problem, it is insensitive to setup costs and an optimal solution cannot be directly obtained from it. However, the solutions can be improved by heuristics and used as upper bounds in the branch-and-price algorithm.

Essentially, the network flow model is a transportation model similar to Horn's model (Horn, 1973). The problem will have *j* origins (one per task) with supplies equal to the processing times, p_j. Let us call each origin T_j. On the other hand, each possible time slot in the schedule will become a destination with demand equal to the number of available machines, *m*. We will call each destination I_t and it refers to the scheduling period $\left[t-1, t\right]$. The index *t* can vary between 1 and t_{max} (which defines the time horizon for the problem). In order to balance the problem, an additional origin must be set to supply the idle time in the schedule. Let that origin be T_L. Its supply is:

$$mt_{max} - \sum_{j=1}^{n} p_j \qquad (3)$$

This value cannot be negative. In that event there would be no feasible solutions, because the processing times would exceed the available capacity.

The following figure represents the proposed model:

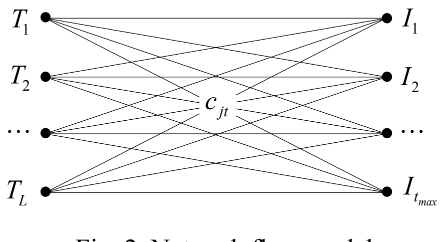

Fig. 2. Network flow model

Note that arc $\left(T_j, I_t\right)$ will only exist if $r_j \geq t-1$. Otherwise, the task is not available for processing at $t-1$. The value c_{jt} refers to the cost of each arc and is computed in the following way:

$$c_{jt} = \begin{cases} 0, & d_j \geq t \\ w_j \left(t - d_j\right), & d_j < t \end{cases} \qquad (4)$$

This computation must agree with the cost function defined in section 1.3 and assures that late work costs will be minimized.

Preposition 1: If all tasks have unit processing times, the solution obtained from the proposed network flow model is optimal.

Proof: If all processing times are unitary, all tasks are scheduled in exactly one scheduling slot, incurring in one setup. The total number of setups will be equal to the number of tasks and it will be optimal.

In the next section we discuss an algorithm to transform a network flow model solution into a valid scheduling solution with minimal number of setups.

2.2 Obtaining a Valid Scheduling Solution

Obtaining a valid scheduling solution to our problem from a network flow solution is trivial. In this section we propose an algorithm that not only achieves a valid solution but also minimizes the number of setups for the given network flow solution.

Let f_{jt} be the flow from T_j to I_t in a given network flow solution. The network flow solution will be completely defined by the following flow matrix, *F*:

$$F = \begin{bmatrix} f_{11} & f_{12} & \cdots & f_{1t_{max}} \\ f_{21} & f_{22} & \cdots & f_{2t_{max}} \\ \cdots & \cdots & \cdots & \cdots \\ f_{n1} & f_{n2} & \cdots & f_{nt_{max}} \end{bmatrix} \qquad (5)$$

Because all values for demand and supply are integers, it is guaranteed that all flow values are also integers. It can easily be shown that, for every time slot, $\sum_{j=1}^{n} f_{jt} \leq m$. In fact, this sum is the number of machines in use for each time slot. Then, for $t=1$ the number of incurred setups is equal to $\sum_{j=1}^{n} f_{j1}$.

On subsequent time slots, for any given machine, setups are not necessary if the scheduled task is the same task scheduled in the previous time slot. In general, at time *t*, we can avoid, at most, a number of setups equal to $\sum_{j=1}^{n} \min\left(f_{jt}, f_{jt+1}\right)$. The proposed algorithm results in a scheduling solution in which all the "avoidable" setups are effectively avoided.

To obtain a solution we first schedule the flows for $t=1$ and proceed chronologically until $t = t_{max}$. For each time slot the F_t vector is decomposed into two vectors: the vector of flows that can be scheduled without setups, F_t^C, and the vector of flows that need setups, F_t^D. Obviously, $F_t = F_t^C + F_t^D$. The two vectors can be calculated as follows:

$$f_{jt}^C = \min\left(f_{jt-1}, f_{jt}\right) \tag{6}$$

$$f_{jt}^D = f_{jt} - f_{jt}^C \tag{7}$$

For the initialization, we must admit $f_{j0} = 0$ for all tasks. This will force $f_{j1}^C = 0$ for all tasks and a setup for each machine used at time $t=1$.

At $t=1$ the choice of machines to schedule flows is irrelevant (a setup will be needed for each used slot). In subsequent periods the flows in vector F_t^C must be scheduled first in machines that have the same task on the previous period. The flows in F_t^D are always scheduled last and the choice of machine is irrelevant.

This algorithm is polynomial with respect to *n*, *m* and t_{max}.

Preposition 2: Applying the algorithm to a given set of flows results in a minimal number of setups for that set of flows.

Proof: For $t=1$ a setup is unavoidable for each used machine. For subsequent periods and for each task, if $f_{jt} \le f_{jt-1}$ no setups are needed for that task. In that case $f_{jt}^C = f_{jt}$ and no setup will be incurred. If $f_{jt} > f_{jt-1}$, at least $f_{jt} - f_{jt-1}$ must be incurred. Now, $f_{jt}^D = f_{jt} - f_{jt-1}$ and only the minimal number of setups is incurred.

In the following section we will present the integer linear programming model used to obtain the optimal solution to the problem.

2.3 Integer Programming Model

As the network flow model presented on the previous sections does not provide an optimal solution to the problem, we developed an integer programming approach to solve the problem. In this section we present the developed model.

The proposed formulation is based on time indexed variables. This kind of variables was first used to solve scheduling problems by Sousa *et al.* (1992). In this context, we use the binary variables x_{ijt} which takes the value 1 if task *j* is scheduled on machine *i* at time slot *t*, which represents the period $[t-1, t]$. Otherwise, these variables will take the value 0.

To account for the number of setups we must also introduce a set of binary variables, y_{ijt}. If setting $x_{ijt} = 1$ a setup is incurred, and the corresponding variable, y_{ijt}, must take the value 1. If $x_{ijt} = 0$ or no setup is incurred, the corresponding *y* variable takes the value 0.

The complete ILP model is as follows:

$$min \sum_{i=1}^{m}\sum_{j=1}^{n}\sum_{t=1}^{t_{max}} y_{ijt} + \sum_{i=1}^{m}\sum_{j=1}^{n}\sum_{t=d_j+1}^{t_{max}} w_j\left(t-d_j\right)x_{ijt} \tag{8}$$

$$s.t. \quad \sum_{i=1}^{m}\sum_{t=1}^{t_{max}} x_{ijt} = p_j, \quad \forall j \tag{9}$$

$$\sum_{j=1}^{n} x_{ijt} \le 1, \quad \forall i, \forall t \tag{10}$$

$$y_{ij1} = x_{ij1}, \quad \forall i, \forall j \tag{11}$$

$$y_{ijt} \ge x_{ijt} - x_{ijt-1}, \quad \forall i, \forall j, \forall t \in \{2, 3, \cdots, t_{max}\} \tag{12}$$

$$x_{ijt} = 0,1, \quad \forall i, \forall j, \forall t \tag{13}$$

$$y_{ijt} = 0,1, \quad \forall i, \forall j, \forall t \tag{14}$$

The objective function (8) follows from section 1.3, in terms of the *x* and *y* variable meanings. The first term accounts for the number of setups and thus, the setup costs. The second term accounts for the late work costs.

Constraints (9) ensure that each task is scheduled in p_j time slots and, therefore, the processing times are met. The next set of constraints (10) ensures that, at most, one task is scheduled at each time slot. Note that they also assure that no more that *m* machines are used in each time unit.

Constraints (11) ensure that a setup is accounted for any task scheduled at time slot 1. Constraints (12) guarantee that, in any machine and in every time slot, if the scheduled task is different from the task scheduled in the previous time slot the appropriate *y* variable takes the value 1.

This problem is not amenable for solution with a commercial optimization package for two reasons: first, the number of variables and constraints rises rapidly with the number of machines, the number of tasks or the planning horizon size; second, the problem is very symmetric, because the same scheduling solution can be implemented with different machine indexes.

For those reasons we implemented a column generation approach to solve the problem that is presented in the next section.

3. COLUMN GENERATION

In this section we start by presenting a Dantzig-Wolfe decomposition of the model and a dynamic programming algorithm for solving the resulting subproblem. Later in the section we propose a branching scheme and we discuss the corresponding implications on the subproblem algorithm.

3.1 Dantzig-Wolfe Decomposition

Consider the mathematical model presented in section 2.3. For simplicity, the following transformations will be performed:

$\mathbf{Z}_i$: a vector of all variables referring to processor *i* (includes all the *x* and all the *y* variables);

$\mathbf{C}_i$: a vector of all objective function coefficients referring to processor *i*;

$\mathbf{A}_i$: a matrix with *j* rows containing all the coefficients referring to machine *i* for the processing times constraints;

$\mathbf{p}$: the vector of processing times;

P_i : a polyhedron defined by all the constraints (except the processing times constraints) that refer to machine *i*.

Using these later definitions, the mathematical model can be rewritten as follows:

$$\begin{aligned} &min && \sum_{i=1}^{m} \mathbf{C}_i^T \mathbf{Z}_i \\ &s.t. && \sum_{i=1}^{m} \mathbf{A}_i \mathbf{Z}_i = \mathbf{p} \\ & && \mathbf{Z}_i \in P_i, \quad \forall i \end{aligned} \tag{15}$$

The set of constraints with a block angular structure was decomposed, leaving the processing times constraints in the master problem and defining a subproblem for each P_i constraint.

3.1.1 The Master Problem

Because P_i is a convex region, any point of that region can be represented as a convex combination of extreme points. Let K_i be the set of extreme points for machine *i* and let P_{k_i} represent an element of that set. This can be translated to the following equality:

$$\mathbf{Z}_i = \sum_{k_i \in K_i} P_{k_i} \lambda_{k_i}, \quad \sum_{k_i \in K_i} \lambda_{k_i} = 1, \quad \lambda_{k_i} \geq 0, \forall k_i \tag{16}$$

After variable substitution the master problem is as follows:

$$\begin{aligned} &min && \sum_{i=1}^{m} \sum_{k_i \in K_i} \left(\mathbf{C}_i^T P_{k_i} \right) \lambda_{k_i} \\ &s.t. && \sum_{i=1}^{m} \sum_{k_i \in K_i} \left(\mathbf{A}_i P_{k_i} \right) \lambda_{k_i} = \mathbf{p} \\ & && \sum_{k_i \in K_i} \lambda_{k_i} = 1, \quad \forall i \end{aligned} \tag{17}$$

Because, in our problem, the processors are identical, the reference to the specific processor can be removed and the convexity constraints can be replaced by their sum:

$$\begin{aligned} &min && \sum_{k \in K} \left(\mathbf{C}^T P_k \right) \lambda_k \\ &s.t. && \sum_{k \in K} \left(\mathbf{A} P_k \right) \lambda_k = \mathbf{p} \\ & && \sum_{k \in K} \lambda_k = m \end{aligned} \tag{18}$$

For simplicity, let $x_k = \lambda_k$, $c_k = \mathbf{C}^T P_k$ and a_{jk} be the elements of $\mathbf{A}P_k$. After the necessary substitutions the master problem can be written as follows:

$$\begin{aligned} &min && \sum_{k \in K} c_k x_k \\ &s.t. && \sum_{k \in K} a_{jk} x_k = p_j, \quad \forall j \\ & && \sum_{k \in K} x_k = m \end{aligned} \tag{19}$$

In this last model a variable can be seen as a schedule for one processor, where c_k is the contribution of that schedule to the global cost and a_{jk} is the amount of processing done for task *j* in that schedule. The collection of all individual machine schedules present in the optimal solution will make up the global schedule. Based on this interpretation the following conclusion can be drawn.

Preposition 3: For a solution to be integral, it is sufficient that all x_k variables have integer values.

Proof: The value of each variable is the number of processors executing the corresponding schedule. Thus, if all variables have integer values the solution can be translated to a valid operational plan.

However, as we will show later, this is not a necessary condition to get a valid solution.

In order to generate new individual machine schedules to include in the master problem, each schedule cost must be determined without knowing in advance the global schedule. As remarked in section 1.3, our objective function complies with this requirement.

It is known that, whenever possible, replacing equality constraints with inequalities leads to better computational performance because the additional restriction of the dual solution space (Desaulniers *et al.*, 2002). As shown next, all the equalities in our master problem can be replaced with inequalities.

Preposition 4: All the constraints in the form $\sum_{k \in K} a_{jk} x_k = p_j$ can be replaced with constraints in the form $\sum_{k \in K} a_{jk} x_k \geq p_j$.

Proof: If a task processing time is exceeded the extra processing can always be removed from the end of the schedule without additional setups or late work.

Preposition 5: The constraint $\sum_{k \in K} x_k = m$ can be replaced by the constraint $\sum_{k \in K} x_k \leq m$.

Proof: If allowed by the processing times, machines can be left idle.

Making these two transformations, the resulting master problem will be:

$$\begin{aligned} &min && \sum_{k \in K} c_k x_k \\ &s.t. && \sum_{k \in K} a_{jk} x_k \geq p_j, \quad \forall j \\ & && \sum_{k \in K} x_k \leq m \end{aligned} \tag{20}$$

3.1.2 The Subproblem

Let π_j (for $j = 1, 2, \cdots, n$) be the set of dual variables associated with the first *n* constraints of the master problem and let π_M be the dual variable associated with the last constraint. To find the column with the maximum reduced cost for the master problem, the following subproblem is solved:

$$\begin{aligned} max \quad & \sum_{j=1}^{n}\sum_{t=1}^{t_{max}} \pi_j x_{jt} + \pi_M - \sum_{j=1}^{n}\sum_{t=1}^{t_{max}} y_{jt} - \sum_{j=1}^{n}\sum_{t=d_j+1}^{t_{max}} w_j\left(t-d_j\right)x_{jt} \\ s.t. \quad & \sum_{j=1}^{n} x_{jt} \le 1, \quad \forall t \\ & y_{j1} = x_{j1}, \quad \forall j \\ & y_{jt} \ge x_{jt} - x_{jt-1}, \quad \forall j, \forall t \in \{2,3,\cdots,t_{max}\} \\ & x_{jt} = 0,1, \quad \forall j, \forall t \\ & y_{jt} = 0,1, \quad \forall j, \forall t \end{aligned} \qquad (21)$$

Note that this problem is still hard to solve with a general ILP software package. However it has a special structure that allows the development of a specialized algorithm presented in the next section.

In the objective function, π_j can be seen as a prize for including each unit of task *j* processing in the schedule. The constant π_M (negative) is the penalty for using the processor pool (independently of the schedule). The rest of the objective function, as before, deals with the penalties incurred for setups and late work.

In the next section a dynamic programming algorithm that takes advantage of the subproblem's structure is presented.

3.2 Dynamic Programming Algorithm for the Subproblem

In order to identify new attractive columns to include in the master problem we developed a dynamic programming algorithm that takes advantage of the subproblem special structure.

The developed algorithm has $t_{max}+1$ stages, corresponding to each integral time instant from 0 to t_{max}, and $n+1$ states, representing the last scheduled task (one of the *n* tasks or none). Note that, because of the release dates, not all states are available at every stage.

Let $F_t(j)$ be the objective function value for state *j* and stage *t*. For simplicity assume that at time $t=0$ (stage 0) the machine is not set to any task, i.e., the machine is at state 0. We can start up the dynamic programming algorithm by setting the initial objective function value:

$$F_0(0) = \pi_M \qquad (22)$$

This means that, if no task is scheduled, the reduced cost of the new column will be π_M. For any stage, the objective function value can be calculated recursively by the following expression:

$$F_t(j) = \max\left(F_{t-1}(l) + \pi_j - w_{tj} - p_{lj}\right), \qquad (23)$$

where $t = 1, 2, \cdots, t_{max}$ and $j = \{0, j : r_j < t\}$ and $l = \{0, j : r_j < t-1\}$. We must also define $\pi_0 = 0$ and the values of w_{tj} and p_{lj}:

$$w_{tj} = \begin{cases} 0, & j = 0 \; or \; t \le d_j \\ w_j\left(t-d_j\right), & j > 0 \; and \; t > d_j \end{cases} \qquad (24)$$

$$p_{lj} = \begin{cases} 0, & j = 0 \; or \; l = j \\ 1, & j > 0 \; and \; l \ne j \end{cases} \qquad (25)$$

These last expressions will account the setup costs (25) and the late work costs (24), as explained.

The iterative solution of the master problem (to obtain a dual vector for the subproblem) and of the subproblem (to obtain new columns for the master problem) until there are no more attractive columns will lead to an optimal solution to the linear relaxation of the master problem. Usually, this optimal solution will not obey the integral requirements of the master problem. To get an optimal integral solution we developed the branching scheme presented in the following section.

3.3 Branching Scheme

Branching on the restricted master problem variables is not a good choice, because it may lead to column regeneration (Barnhart, 1998). We do not branch on those variables, but rather on the variables of the network flow model.

Previously we defined f_{jt} as the network flow model variable representing the flow in arc $\left(T_j, I_t\right)$. This variable represents the number of processors allotted to task *j* at time slot *t*. A relation between the flow variables and the restricted master problem can be established by the following expression:

$$f_{jt} = \sum_{k \in K} \delta_{jtk} x_k \qquad (26)$$

The δ_{jtk} coefficients are equal to 1 if, in processor schedule *k*, the *j* task is scheduled at time slot *t*, and are equal to 0 otherwise. These coefficients must be determined at column generation because they cannot be calculated based on the master problem technological coefficients.

Preposition 6: A solution is integer *iff* all f_{jt} are integer.

Proof: The condition is sufficient because, on any integral set of flows, we can apply the algorithm described in section 2.2 to obtain a valid scheduling solution. The condition is necessary because any value of f_{jt} represents the number of machines allotted to task *j* at time slot *t*, which must be integer.

Note that, in order to obtain a valid integer solution, it is not necessary that all x_k are integer. If the resulting flows are integer it is possible to obtain a valid scheduling solution.

If a solution has non-integer flows, one of those flows must be chosen for branching. Because this choice may be computationally relevant, a method for choosing the flows to branch on may be needed. Let us denote the value of the non-integral flow to branch on by f_{jt}^*. The current problem (corresponding to a branch tree node) will originate two child problems with a mutually exclusive set of solutions: in one of these problems the constraint $f_{jt} \leq \lfloor f_{jt}^* \rfloor$ is added; in the other problem we add the constraint $f_{jt} \geq \lceil f_{jt}^* \rceil$. As a result, a set of invalid solutions is removed from the solution space.

3.3.1 Subproblem Consequences

Adding new constraints to the master problem modifies its structure. For the subproblem to identify new attractive columns for the modified master problem, those modifications must be accounted for in the subproblem. Basically, the dual information associated with the new constraints must be considered at the subproblem level.

Every extra constraint respects to the scheduling of a specific task in a specific time slot. The dual value associated to an extra constraint can be seen as a prize or penalty for scheduling such task in such time slot. Therefore, the modifications to the subproblem can be easily done because they only affects the calculation of the state transition costs in the dynamic programming formulation.

Consider a node in the search tree. At that node, the set of added constraints can be divided into two subsets: a subset *G* of constraints in the form

$$\sum_{k\in K} \delta_{jtk} x_k \leq f_g, \tag{27}$$

for every $g \in G$; and a subset *H* of constraints in the form

$$\sum_{k\in K} \delta_{jtk} x_k \geq f_h, \tag{28}$$

for every $h \in H$.

Let us denote each dual variable associated with a subset *G* constraints by μ_g and each dual variable associated with constraints from subset *H* by ν_h.

The new dynamic programming recurrence equation can be written as follows:

$$F_t(j) = \max\left(F_{t-1}(l) + \pi_j + \sum_{g\in G_{jt}} \mu_g + \sum_{h\in H_{jt}} \nu_h - w_{tj} - p_{lj} \right), \tag{29}$$

where G_{jt} and H_{jt} are subsets containing only the constraints referring to task *j* and time slot *t*. It can be easily seen that the only modification involves adding the values of the dual variables associated with a specific task and a specific time slot.

4. CONCLUSIONS AND RESULTS

In this work we presented an algorithm for solving exactly a scheduling problem with identical parallel machines and malleable tasks, subject to arbitrary release dates and due dates in order to minimize a function of the late work and number of setups. To our knowledge, this kind of problem has never been tackled with column generation and branch-and-price before.

To get an idea about the size of instances we could solve, we made a computational implementation of the algorithm using CPLEX and generated some random instances. We concluded that the number of tasks was determinant for the solution times. With about 10 tasks, we consistently solved problems with 40 machines and about 20 time slots. Lowering one of these values allowed for the others to be increased.

Acknowledgements
This work was supported by the European Union through the European Social Found and the Portuguese project Prodep III.

5. REFERENCES

Barnhart, C., Johnson, L., Nemhauser, G.L., Savelsbergh, M.W. and Vance, P.H. (1998). Branch-and-Price: Column Generation for Solving Huge Integer Problems, *Operations Research*, **46**, 316-329.

Blazewicz, J., Kovalyov, M.Y., Machowiak, M., Trystram, D. and Weglarz, J. (2004). Scheduling Malleable Tasks on Parallel Processors to Minimize the Makespan, *Annals of Operations Research*, **129**, 65–80.

Desaulniers, G., Desrosiers, J. and Solomon, M.M. (2002). Accelerating strategies in column generation methods for vehicle routing and crew scheduling problems. In: Ribeiro, C. and Hansen, P. (eds.) *Essays and Surveys in Metaheuristics*. Norwell, MA: Kluwer, 309-324.

Horn, W.A. (1973). Minimizing Average Flow Time with Parallel Machines, *Operations Research*, **21**, 846-847.

Sousa, J.P., Wolsey, L.A. (1992). A time indexed formulation of non-preemptive single machine scheduling problems, *Mathematical Programming*, **54**, 353-367.