Smoothing the lagrangean dual function when the subproblem is solved through dynamic programming

Artur Alves Pessoa
Departamento de Engenharia de Produção - Universidade Federal Fluminense
Rua Passos da Pátria, 156, São Domingos, 24210-240, Niterói, RJ
artur@producao.uff.br

Marcos Costa Roboredo
Departamento de Engenharia de Produção - Universidade Federal Fluminense
Rua Passos da Pátria, 156, São Domingos, 24210-240, Niterói, RJ
mcr.marcos@yahoo.com.br

RESUMO
Ao utilizar relaxações lagrangeanas para resolver problemas de otimização combinatória, a função dual lagrangeana resultante não é diferenciável, o que pode gerar grandes dificuldades de convergência. Neste artigo, introduzimos uma nova técnica de suavização para funções duais lagrangeanas associadas a problemas de otimização combinatória quando o subproblema é resolvido por programação dinâmica. Nossa técnica permite a utilização de métodos de gradiente de subida para aproximar os valores ótimos dos multiplicadores de lagrange para o problema original. Também provamos que, quando o gradiente obtido é zero, uma solução fracionária viável para o problema original é dada. Utilizamos esta técnica para calcular limites inferiores para o Problema de Atribuição Generalizado. Para este problema, nossos experimentos mostram que limites inferiores muito próximos aos ótimos podem ser obtidos em um tempo computacional modesto quando comparado ao melhor algoritmo de geração de colunas estabilizada encontrado na literatura.

PALAVRAS CHAVE. Otimização, Relaxação Lagrangeana, Métodos de gradiente, programação dinâmica.

ABSTRACT
When lagrangean relaxation is applied to combinatorial optimization problems, the resulting dual function becomes non-differentiable, which may generate serious convergence difficulties. In this paper, we introduce a new technique for smoothing the lagrangean dual function associated to relaxed combinatorial optimization problems when the subproblem is solved via dynamic programing. Our technique allows for using gradient ascent methods to approach the optimal values of the lagrangean multipliers for the original problem. We also prove that when the obtained gradient is zero, a feasible fractional solution for the original problem is given. We use this technique to calculate lower bounds for the classical Generalized Assignment Problem. For this problem, our experiments show that lower bounds very close to the optimal ones can be found in a modest computational time, when compared to the best stabilized column generation algorithm found in the literature.

KEY WORDS. Optimization, Lagrangean Relaxation, gradient methods, dynamic programming.
1 Introduction

Lagrangean Relaxation is a widely used technique to generate tight lower bounds for optimization problems. This technique consists in solving a sequence of subproblems varying the values of lagrangean multipliers. Upon every subproblem resolution, a new lower bound on the original problem is generated. The function that maps each vector of multiplier values to the corresponding lower bound is called dual lagrangean function. In combinatorial optimization problems, the dual lagrangean function is not differentiable at many points, and, in many cases, the subproblem is solved through dynamic programming (Bellman, 1966). Among these problems, we highlight the following ones: The GAP (Generalized Assignment Problem) (Pigatti et al., 2005), Scheduling in one machine minimizing the total weighted tardiness \(1\|\sum w_j T_j\) (Rodrigues et al., 2008), GQAP(Generalized Quadratic Assignment Problem) (Pessoa et al., 2008), etc. Since the dual function is not differentiable, we can not apply the most efficient non-linear programming methods such as gradient descent methods (Hestenes, 1969), conjugate gradient methods (Hestenes and Stiefel, 1952), among others.

In this paper, we introduce a new technique for smoothing the lagrangean dual function associated to relaxed combinatorial optimization problems when the subproblem is solved via dynamic programming. The resulting function is concave, differentiable in all its domain and approaches the original function from below with a controlled error. As consequence, our technique allows for using gradient ascent methods to approach the optimal values of the lagrangean multipliers for the original problem. The gradient of our smooth function is calculated through a modified dynamic programming algorithm. During this calculation, one also obtains a fractional solution that can be written as a convex combination of feasible solutions for the original subproblem. We prove that the obtained gradient is equal to the vector of constraint violations for this fractional solution. As a result, finding a vector of lagrangean multipliers that leads to the gradient zero leads to an associated feasible fractional solution for the original problem. For the sake of simplicity, we present our technique and theoretical and experimental results in the context of the GAP. For this problem, our experiments show that lower bounds very close to the optimal ones can be found in a modest computational time, when compared to the best column generation algorithm found in the literature.

In order to illustrate our technique, let us consider the following toy combinatorial optimization problem. Although the subproblem solution does not necessarily involve dynamic programming, it is useful for our purposes. the problem has only two possible solutions: \(s_1 \in s_2\). Each of the two solutions has a cost and a resource usage, where \(\text{cost}(s_1) = 2\), \(\text{cost}(s_2) = 10\), \(\text{usage}(s_1) = 35\), and \(\text{usage}(s_2) = 15\). Our aim is to minimize the cost, with the constraint that the resource usage does not exceed 20 units. The formulation of the problem is:

\[
\begin{align*}
\text{Minimize} & \quad \text{cost}(x) \\
\text{subject to} & \quad \text{usage}(x) \leq 20 \\
& \quad x \in \{s_1, s_2\}
\end{align*}
\]

(1)

We apply the lagrangean relaxation method to this problem dualizing the constraint (1). Let \(\theta\) be the corresponding lagrangean multiplier. The lagrangean subproblem becomes: Minimize \(L(x, \theta) = \text{cost}(x) + \theta(\text{usage}(x) - 20)\), subject to \(x \in \{s_1, s_2\}\). The lagrangean dual problem is: Maximize \(L(\theta)\), subject to \(\theta \geq 0\). If \(x\) is an optimal solution of the lagrangean subproblem, the lagrangean dual function \(L(\theta) = L(x, \theta)\). It is easy to note that: \(L(s_1, \theta) = 2 + 15\theta\), \(L(s_2, \theta) = 10 - 5\theta\), and \(L(\theta) = \text{min}\{2 + 15\theta, 10 - 5\theta\}\).
Observe that \( L(\theta) \) is not differentiable only because the min function (this is also the case when the subproblem is solved via dynamic programming). Therefore, if we replace this function by an approximate smooth function, we also obtain a smooth function as the lagrangean dual. In this paper the min function is replaced by an hyperbolic function that approaches the former from bellow with an error \( \delta \), for any value of \( \delta > 0 \). Figure 1 represents the lagrangean dual function for the previous toy problem and the corresponding approximate smooth functions for three values of \( \delta \). The function \( L(\theta) \) and its three approximations with errors \( \delta = 1 \), \( \delta = 2 \), and \( \delta = 4 \) are denoted by \( L \), \( L_1 \), \( L_2 \) and \( L_4 \) respectively.

Figure 1: The lagrangean dual function for the toy problem and the corresponding approximate smooth functions for \( \delta = 1 \), 2, and 4.

Note that the more the value of \( \delta \) decreases, the more the maximum value of the smooth function approaches the maximum value of the original dual function.

The choice of the GAP to illustrate the method is due to the fact it is a classical combinatorial optimization problem for which a suitable lagrangean relaxation is known to give strong lower bounds but having serious convergence problems when using traditional methods such as column generation or subgradient search. Motivated by this fact, Fisher et al. (1986) proposed a special multiplier adjustment method (MAM) for this problem that explores the fact that the subproblem is solved via dynamic programming. This method, which has been later improved by Guignard and Rosenwein (1989), consists of successively finding the smallest dual step that changes the subproblem solution. In the opposite of what our method does, the MAM walks through the points where the lagrangean dual function is not differentiable. On the other hand, one can argue that both methods use the dynamic programming technique to extract information from more than one subproblem solution at each iteration. Recently, Pigatti et al. (2005) use a branch-and-price algorithm to exactly solve several GAP instances from the literature. In order to calculate the lower bounds, they use a stabilized column generation approach. More recently, Avella et al. (2008) proposed a cut-and-branch algorithm for the GAP based on a procedure for exact knapsack separation. Instead of using lagrangean relaxation, they start with a weaker formulation and apply all violated cuts that can be derived from knapsack subproblems. The obtained lower bound is equivalent to that of the lagrangean relaxation. Since the aim of this paper is testing the newly proposed technique for solving lagrangean dual problems, we do not compare our method against the method of Avella et al.

Moreno et al. (2008) used the same smoothing function to approximately solve optimization problems where the objective function is non-differentiable. This work solves a particular case of the prob-
lems addressed by Moreno et al. (2008), since our (dual) objective function is concave and piecewise linear. On the other hand, we are able to handle functions with an exponentially large number of linear pieces since they are evaluated via dynamic programming. Moreover, we give an interesting relation between the gradients of this function and the primal solutions of the original problem.

This paper is organized as follows. In Section 2, we define the GAP and show a lagrangean relaxation formulation for it. In Section 3, we mathematically define our smooth approximation for the lagrangean dual function and the resulting recursion for solving the modified subproblem via dynamic programming. In Section 4, we briefly describe the gradient ascent method that we use in our experiments. Later, in section 5, we prove the convergence of this method and show that a fractional feasible solution is obtained at the end of its execution. Finally, in section 6, we report the results of our computational experiments.

2 The Generalized Assignment Problem

The GAP is defined mathematically as follows: Let \( I = \{1, \ldots, m\} \) be a set of machines and \( J = \{1, \ldots, n\} \) a set of jobs. For each machine \( i \in I \), we are given the available time \( b_i \) of machine \( i \), for each machine \( i \in I \) and each job \( j \in J \), we are given both the cost \( c_{ij} \) of assigning the job \( j \) to the machine \( i \), and the time \( a_{ij} \) required to process the job \( j \) in the machine \( i \). Our aim is to assign each job to exactly one machine so that the total processing time assigned to each machine does not exceed its available time, and minimizing the total assignment cost.

Bellow, we give a basic integer programming formulation for the GAP that uses binary variables \( x_{ij} \), indicating that the job \( j \) is assigned to the machine \( i \), for each \( i \in I \) and \( j \in J \).

\[
\begin{align*}
\min & \quad \sum_{j=1}^{n} \sum_{i=1}^{m} c_{ij} x_{ij} \\
\text{subject to} & \quad \sum_{j=1}^{n} a_{ij} x_{ij} \leq b_i \quad i \in I \\
& \quad \sum_{i=1}^{m} x_{ij} = 1 \quad j \in J \\
& \quad x_{ij} \in \{0, 1\} \quad i \in I, j \in J.
\end{align*}
\]

In the lagrangean relaxation used in this paper (and also used by Fisher et al. (1986); Guignard and Rosenwein (1989); Pigatti et al. (2005)), we dualize the constraints (4) using the multipliers \( \pi_j \), for all \( j \in J \). This leads to the following lagrangean subproblem:

\[
\begin{align*}
[L(\pi)] \quad & \min \sum_{j=1}^{n} \sum_{i=1}^{m} c_{ij} x_{ij} + \sum_{j=1}^{n} \pi_j (\sum_{i=1}^{m} x_{ij} - 1) \\
\text{subject to} & \quad \sum_{j=1}^{n} a_{ij} x_{ij} \leq b_i \quad i \in I \\
& \quad x_{ij} \in \{0, 1\} \quad i \in I, j \in J.
\end{align*}
\]

Rewriting (6), we obtain \([L(\pi)] \) equivalent to:
\[
\min \sum_{i=1}^{m} \sum_{j=1}^{n} (c_{ij} + \pi_j) x_{ij} - \sum_{j=1}^{n} \pi_j
\]

subject to (7) e (8).

The corresponding lagrangean dual problem is:

\[ [LD] \max_{\pi} V \left( L(\pi) \right) \],

where \( V[X] \) denotes the value of the optimal solution subproblem \( X \).

Following the lagrangean decomposition approach of Guignard and Kim (1987), we can to decompose \([L(\pi)]\) into \( m \) subproblems. For each \( i \in I \), we solve:

\[ [KP(\pi)^i] \min \sum_{j=1}^{n} (c_{ij} + \pi_j) x_{ij} \]

subject to (7) and (8).

Each subproblem above is a boolean knapsack problem \((KP)\) (Kellerer et al., 2004) and can be solved by dynamic programming in \( O(b_i n) \) time, with the usual recursion:

\[
OPT(l, k) = \begin{cases} 
\min\{OPT(l - 1, k), OPT(l - 1, k - a_{il}) + c_{ij} + \pi_j\}, & \text{if } k \geq a_{il} \text{ and } l > 1, \\
OPT(l - 1, k), & \text{if } k < a_{il} \text{ and } l > 1, \\
\min\{0, c_{ij} + \pi_j\}, & \text{if } k \geq a_{il} \text{ and } l = 1, \\
0, & \text{if } k < a_{il} \text{ and } l = 1,
\end{cases}
\]

for all \( l \in N \) and \( k \in \{0, ..., b_i\} \), where \( OPT(l, k) \) denotes the cost of a knapsack problem considering only the items 1, ..., \( l \) and capacity \( k \).

As a result, we have \( V[KP(\pi)^i] = OPT_i(n, b_i), \forall i \in I \). Then, solving the lagrangian subproblem \([L(\pi)]\) reduces to solving \( m \) \( KPs \), giving the following lower bound on the original problem:

\[
V \left[ L(\pi) \right] = \sum_{i=1}^{m} V \left[ KP(\pi)^i \right] - \sum_{j=1}^{n} \pi_j, \text{ or equivalently,}
\]

\[
V \left[ L(\pi) \right] = \sum_{i=1}^{m} OPT_i(n, b_i) - \sum_{j=1}^{n} \pi_j.
\]

### 3 The Smooth Lagrangian Dual Function

By the recursion (10), we notice that the lagrangean dual function \( V[L(\pi)] \) is not differentiable only due to the function \( f(x, y) = \min\{x, y\} \). In this section, we formalize a smooth approximation for \( f \) in order to obtain a smooth approximation for \( V[L(\pi)] \). We denote the approximation of \( f \) by \( \tilde{f} \) that is defined as follows: \( \tilde{f}(x, y) = \frac{x+y-\sqrt{(x-y)^2+4\delta^2}}{2} \), where \( \delta \) is the approximation error.

The approximation function \( \tilde{f} \) has the following suitable properties:
• $\tilde{f}$ is concave.
• $f(x, y) - \delta < \tilde{f}(x, y) < f(x, y)$.
• $\lim_{(x-y) \to +\infty} \tilde{f}(x, y) = y$.
• $\lim_{(x-y) \to -\infty} \tilde{f}(x, y) = x$.
• $\frac{\partial \tilde{f}(x,y)}{\partial x} = \frac{1}{2} - \frac{2(x-y)}{4\sqrt{(x-y)^2 + 4\delta^2}} = \frac{1}{2} \left( 1 - \frac{x-y}{\sqrt{(x-y)^2 + 4\delta^2}} \right)$.
• $x = y \Rightarrow \frac{\partial \tilde{f}(x,y)}{\partial x} = \frac{1}{2}$.
• $\frac{\partial \tilde{f}(x,y)}{\partial x} + \frac{\partial \tilde{f}(x,y)}{\partial y} = 1 - \left( \frac{x-y+y-x}{\sqrt{(x-y)^2 + 4\delta^2}} \right) = 1$.

Most the properties above have immediate proofs that we omit in this paper. Below, we prove the first one.

**Theorem 1** $\tilde{f}$ is concave.

**Proof:** To prove $\tilde{f}$ is concave, it suffices to prove that $g(z) = \sqrt{z^2 + 4\delta^2}$ is convex. This is true because one can easily check that $g''(z) > 0$ for all $z$. ■

As a result of smoothing $\tilde{f}$, we obtain the following new approximation for recursion (10):

$$
\tilde{OPT}(l, k) = \begin{cases} 
\tilde{f}(\tilde{OPT}(l - 1, k), \tilde{OPT}(l - 1, k - a_{il}) + c_{ij} + \pi_j), & \text{if } k \geq a_{il} \text{ and } l > 1, \\
\tilde{OPT}(l - 1, k), & \text{if } k < a_{il} \text{ and } l > 1, \\
\tilde{f}(0, c_{ij} + \pi_j), & \text{if } k \geq a_{il} \text{ and } l = 1, \\
0, & \text{if } k < a_{il} \text{ and } l = 1,
\end{cases}
$$

(12)

for all $l \in \mathbb{N}$ and $k \in \{0, ..., b_i\}$, where $\tilde{OPT}(l, k)$ denotes the approximated cost of a knapsack problem considering only the items $1, ..., l$ and capacity $k$.

Thus, the final bound (11) is approached by:

$$
V \left[ \tilde{L}(\pi) \right] = \sum_{i=1}^{m} \tilde{OPT}_i(n, b_i) - \sum_{j=1}^{n} \pi_j.
$$

From now on, for the sake of easy notation, we replace the cost of dual problem $V \left[ \tilde{L}(\pi) \right]$ by simply $\tilde{L}(\pi)$. 
4 The gradient method to adjust the multiplier

In this section, we describe the gradient ascent method that we use to adjust the lagrangean multipliers of (9). This method is an approximation of the gradient steepest ascent method (Ben-Tal and Nemirovski, 2004), that, at each iteration, takes the gradient direction and finds the step size in this direction that leads to the largest value of the function to be maximized.

Let \( \pi^k \in \mathbb{R}^n \) be the vector of lagrangean multipliers obtained at the iteration \( k \). At the iteration \( k + 1 \), the steps to determine \( \pi^{k+1} \) are the following:

**Step 1:** Calculate \( d_k = \nabla \tilde{L}(\pi^k)/||\nabla \tilde{L}(\pi^k)||. \)
**Step 2:** Find the integer number \( r \) that minimizes \( \tilde{L}(\pi^k + r\eta \lambda_k d_k) \). then \( \lambda_{k+1} \leftarrow r\eta \lambda_k. \)
**Step 3:** \( \pi^{k+1} \leftarrow \pi^k + \lambda_{k+1} d_k. \)

The steps above are repeated until \( ||\nabla \tilde{L}(\pi^k)|| \) is smaller than a given parameter \( \epsilon \), where \( \lambda_0 \) (the initial step size) and \( \eta \) are input parameters, with \( \lambda_0 > 0 \) and \( \eta > 1 \). Each repetition of the steps above are referred to as major iterations while the iterations required to find the correct value of \( r \) in the Step 2 are called the minor iterations.

Next, we show how to calculate \( \nabla \tilde{L}(\pi) \). For that, we need the following notations:

1. \( S^i_{q,k} \) — the set of all subsets \( Q \) of \( J_q = \{ q, ..., n \} \) whose sum \( \sum_{l \in Q} a_{il} = k \), where \( J_{n+1} = \emptyset. \)
2. \( k_i(Q,l) = \sum_{p \in Q \cap \{1, ..., n \}} a_{lp} \)
3. \( k_{il}(x,y) = \partial f(x,y)/\partial x. \)
4. \( k_{iy}(x,y) = \partial f(x,y)/\partial y. \)
5. \( h_{x,l}^{i,k} = k_{il}(\text{OPT}_i(l-1,b_i-k),\text{OPT}_i(l-1, b_i - k - a_{il}) + c_{il} + \pi_i) \) if \( l \leq b_i. \)
6. \( h_{y,l}^{i,k} = k_{il}(\text{OPT}_i(l-1, b_i-k),\text{OPT}_i(l-1, b_i - k - a_{il}) + c_{il} + \pi_i) \) if \( l \leq b_i. \)
7. \( h_{x,l}^{i,k} = h_{y,l}^{i,k} = 0 \) if \( k > b_i. \)

In order to compute \( \nabla \tilde{L}(\pi^k) \), we use an auxiliary matrix having the same dimensions as the matrix \( \text{OPT}_i \). This matrix, denoted by \( A U X_i \), also gives a fractional solution associated with \( \text{OPT}_i(n,b_i). \) Each cell \( A U X_i(l,k) \) represents the fraction of the partial solution \( \text{OPT}_i(l,k) \) to be used in the final solution. Having this definition in mind, it is easy to see that:

\[
A U X_i(n,k) = \begin{cases} 
1 & \text{if } k = b_i, \\
0 & \text{if } k \neq b_i. 
\end{cases}
\]

In other words, \( \text{OPT}_i(n,b_i) \) is used entirely in the final solution. For all \( l \in \{0, ..., n-1\}, k \in \{0, ..., b_i\} \), we define: \( A U X_i(l, b_i - k) = h_{x,l}^{i,k} \times A U X_i(l + 1, b_i - k) + h_{y,l}^{i,k} \times A U X_i(l + 1, b_i - k + a_{l+1}) \) where \( A U X_i(l,k) \) is defined as zero for all \( l \in \{1, ..., n\} \) and \( k > b_i. \)

It is easy to see that the matrix \( A U X_i \) can also be filled in \( O(b_i n) \) time. Later, we show that, for all \( j \in J \), the fraction of the item \( j \) that is placed in the knapsack \( i \) is given by:

\[
x_{ij} = \sum_{k=0}^{b_i} A U X_i(j, b_i - k) \times h_{y,j}^{i,k}. 
\]
The theorem below show that each component of the gradient vector is equal to the violation of the associated dualized constraint. Therefore, if the gradient is zero, we have a fractional feasible solution.

**Theorem 2** \( \nabla \tilde{L} (\pi) = v \), where \( v_j = \sum_{i=1}^{m} x_{ij} - 1, \forall j \in J \).

**Proof:** We omit this proof due to the lack of space.

## 5 Convergence Proof

We have already shown how to calculate \( \nabla \tilde{L} (\pi) \), and that \( \nabla \tilde{L} (\pi) = 0 \) implies that the associated fractional solution is feasible. Now, we show that the sequence of gradients found upon each major iteration of our method converges to zero. For the GAP, it may happens that the set of optimal lagrangean multipliers (for the original lagrangean dual function) is unbounded. In this case, a point of gradient zero may not exist. Hence, in order to prove the convergence of our method, we use the following two conditions on the GAP instance.

**Condition 1** For all \( j \in N \), there is a subproblem solution where each job is processed in exactly one machine, except for the job \( j \), which is processed in two machines.

**Condition 2** For all \( j \in N \), there is a subproblem solution where each job is processed in exactly one machine, except for the job \( j \), which is not processed in any machine.

Note that the Condition 2 is always satisfied by feasible instances. Although this is not necessarily true for Condition 1, we checked that this condition is satisfied by all benchmark instances that we have tested in our experiments (taken from the widely used OR-Library). The following theorem prove the convergence of our method under these conditions.

**Theorem 3** Let \( X^{**} = \{ \pi \in \mathbb{R}^n | \nabla \tilde{L} (\pi) = 0 \} \). For the gradient steepest ascent method, the following propositions are valid:

(i) If the trajectory \( \{ \pi_t \} \) of the method is limited, then the trajectory has points of accumulation, and all these points of accumulation are critical points of \( \tilde{f} \).

(ii) If the set \( S = \{ \pi \in \mathbb{R}^n | \tilde{L} (\pi) \geq \tilde{L} (\pi^0) \} \) is limited, then the trajectory of the method is limited (and therefore all accumulation points of \( S \), by (i), belong to \( X^{**} \)).

**Proof:** See (Ben-Tal and Nemirovski, 2004)

**Theorem 4** The method proposed in this paper converges.

**Proof:** Note that the step sizes of our method approximates the step sizes of the gradient steepest ascent method by a constant factor. This approximation does not change the accumulation points. Using the previous theorem, it suffices to prove that the trajectory of the proposed method is limited. For this, we need two comments:

For each solution of Condition 1, consider the function \( \tilde{L}^+_j (\pi) = p_j - \pi_j \), where \( p_j \) is a constant. For each solution of Condition 2, consider the function \( \tilde{L}^-_j (\pi) = q_j + \pi_j \), where \( q_j \) is a constant.
Note that, for any \( j \in N \) and any \( \pi \in \mathbb{R}^n \), we have both \( L(\pi) \leq \tilde{L}_j^+(\pi) \) and \( L(\pi) \leq \tilde{L}_j^-(\pi) \). We use this fact to demonstrate that \( \tilde{L}_j^+ \) and \( \tilde{L}_j^- \) limit the trajectory of the method. For that, let \( \pi^t \) be the multiplier on iteration \( t \). For any \( k \in N \), there exists \( \pi^* \in \mathbb{R} \) such that \( \tilde{L}_k^+(\pi^*) = L(\pi^0) \). Thus, we have \( \pi^*_k \leq \pi^* \), otherwise we would have \( L(\pi^t) < L(\pi^0) \), which is not possible since our method is a gradient ascent method.

Hence, in the trajectory of our method \( \pi_k \) has an upper bound, for any \( k \in N \). Similarly, using \( \tilde{L}_k^- \) we can show that \( \pi_k \) has also a lower bound, for any \( k \in N \), what concludes our proof. ■

It is interesting to note that, since the dual function is concave (see theorem 1), the fractional feasible solution found is associated to the optimal value of the smoothed lagrangean dual function.

6 Computational results

Our method was tested on several GAP instances found in the OR-Library. There are 5 types of instances in OR-Library: (A, B, C, D and E), designated as the example: d05100 is an instance of type D with 5 machines and 100 tasks.

We use \( \eta = 2 \) and \( \lambda_0 = 1 \) as parameters of our method. In addition, we start with \( \delta = 5 \) and multiply the value of \( \delta \) by 0.7 whenever \( ||\nabla \tilde{L}(\pi)|| < \delta \).

Table 1 shows some statistics of our method execution, where we used two different stopping conditions: \( \nabla \tilde{L}(\pi) < 0.2 \) and \( \nabla \tilde{L}(\pi) < 0.02 \). For each instance, the two rows show the results for those stopping conditions in that order. The following headers are used for the columns:

- **Best UB**: Best known upper bounds on the optimal value for the GAP.
- **\( \tilde{L}(\pi) \)**: Final value of the smoothed function.
- **LB**: Lower bound obtained when solving the original lagrangean subproblem using the final lagrangean multipliers.
- \( ||\nabla \tilde{L}(\pi)|| \): Final value of \( ||\nabla \tilde{L}(\pi)|| \).
- **# Major**: Number of major iterations performed.
- **# Major**: Number of minor iterations performed.
- **\( C_f \)**: Cost of the fractional solution associated with the current multiplier.
- **Time**: Total execution time.
- **gap (%)**: Relative difference between LB and Best UB, given by \( 100(\text{Best UB} - \text{LB})/\text{Best UB} \).

Table 2 shows the comparison between our method and the column generation algorithm proposed by Pigatti et al. (2005). This method uses the continuous relaxation of (2)-(5) as a hot start for the column generation. The time required to solve this relaxation is negligible when compared to the overall time. Hence, we report this lower bound in the column labeled by Hot Start. The printed bold values in Table 2 indicate the smallest numbers of iterations and times for each instance. We also marked the lower bound values obtained by our method which are equal to those obtained by Pigatti et al. (2005) (the optimal ones). We aborted the executions of the methods in the cases where their running times
Table 1: Smooth dual optimization statistics

<table>
<thead>
<tr>
<th>Instance</th>
<th>Best UB</th>
<th>$\tilde{\pi}$</th>
<th>$\tilde{L}$</th>
<th>$\nabla \tilde{L}(\pi)$</th>
<th>$C_f$</th>
<th># Major</th>
<th># Minor</th>
<th>Time gap (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>c10400</td>
<td>5597</td>
<td>5576</td>
<td>5594</td>
<td>0.196</td>
<td>5669</td>
<td>52</td>
<td>238</td>
<td>115.2</td>
</tr>
<tr>
<td>d10400</td>
<td>24969</td>
<td>24927</td>
<td>24958</td>
<td>0.123</td>
<td>24817</td>
<td>38</td>
<td>168</td>
<td>273.9</td>
</tr>
<tr>
<td>e10400</td>
<td>45746</td>
<td>45728</td>
<td>45743</td>
<td>0.164</td>
<td>45272</td>
<td>146</td>
<td>754</td>
<td>1172.1</td>
</tr>
<tr>
<td>c20400</td>
<td>4782</td>
<td>4745</td>
<td>4777</td>
<td>0.196</td>
<td>4849</td>
<td>60</td>
<td>278</td>
<td>123.4</td>
</tr>
<tr>
<td>d20400</td>
<td>24417</td>
<td>24262</td>
<td>24340</td>
<td>0.114</td>
<td>24301</td>
<td>50</td>
<td>241</td>
<td>314.9</td>
</tr>
<tr>
<td>e20400</td>
<td>44574</td>
<td>44555</td>
<td>44557</td>
<td>0.199</td>
<td>43905</td>
<td>212</td>
<td>1136</td>
<td>1426.1</td>
</tr>
<tr>
<td>c40400</td>
<td>55414</td>
<td>55389</td>
<td>55401</td>
<td>0.155</td>
<td>55694</td>
<td>44</td>
<td>1980</td>
<td>8866.5</td>
</tr>
<tr>
<td>d40400</td>
<td>11341</td>
<td>11272</td>
<td>11337</td>
<td>0.161</td>
<td>11456</td>
<td>54</td>
<td>248</td>
<td>641.3</td>
</tr>
<tr>
<td>e40400</td>
<td>102422</td>
<td>102373</td>
<td>102418</td>
<td>0.195</td>
<td>101779</td>
<td>121</td>
<td>638</td>
<td>1003.1</td>
</tr>
<tr>
<td>c60900</td>
<td>9328</td>
<td>9320</td>
<td>9324</td>
<td>0.183</td>
<td>9358</td>
<td>60</td>
<td>294</td>
<td>686.8</td>
</tr>
<tr>
<td>d60900</td>
<td>54606</td>
<td>54468</td>
<td>54486</td>
<td>0.139</td>
<td>54573</td>
<td>59</td>
<td>295</td>
<td>2410.5</td>
</tr>
<tr>
<td>e60900</td>
<td>100169</td>
<td>100017</td>
<td>100135</td>
<td>0.183</td>
<td>101397</td>
<td>172</td>
<td>925</td>
<td>1424.0</td>
</tr>
<tr>
<td>c801600</td>
<td>9328</td>
<td>9320</td>
<td>9324</td>
<td>0.183</td>
<td>9358</td>
<td>60</td>
<td>294</td>
<td>686.8</td>
</tr>
<tr>
<td>d801600</td>
<td>54606</td>
<td>54468</td>
<td>54486</td>
<td>0.139</td>
<td>54573</td>
<td>59</td>
<td>295</td>
<td>2410.5</td>
</tr>
<tr>
<td>e801600</td>
<td>100169</td>
<td>100017</td>
<td>100135</td>
<td>0.183</td>
<td>101397</td>
<td>172</td>
<td>925</td>
<td>1424.0</td>
</tr>
</tbody>
</table>

exceeded 100000 seconds. We did not report results for instances with $n = 1600$ because the column generation takes too much time on these instances.

Observing the results of Table 2, we notice that the proposed method is faster than the column
Table 2: Comparison against column generation

<table>
<thead>
<tr>
<th>Instance</th>
<th>Hot</th>
<th>Start</th>
<th>Best known</th>
<th>This paper</th>
<th>Pigatti et al.</th>
</tr>
</thead>
<tbody>
<tr>
<td>c10400</td>
<td>5592</td>
<td>5597</td>
<td>5596</td>
<td>1723</td>
<td>778.1</td>
</tr>
<tr>
<td>d10400</td>
<td>24956</td>
<td>24969</td>
<td>24959</td>
<td>754</td>
<td>1172.1</td>
</tr>
<tr>
<td>c10400</td>
<td>45740</td>
<td>45746</td>
<td>45745</td>
<td>2293</td>
<td>659.6</td>
</tr>
<tr>
<td>e10400</td>
<td>4275</td>
<td>4279</td>
<td>4244</td>
<td>4244</td>
<td>1723</td>
</tr>
<tr>
<td>c20400</td>
<td>44862</td>
<td>44877</td>
<td>44876</td>
<td>5651</td>
<td>1457.7</td>
</tr>
<tr>
<td>d20400</td>
<td>24553</td>
<td>24585</td>
<td>24560</td>
<td>1025</td>
<td>1455.3</td>
</tr>
<tr>
<td>c30900</td>
<td>9975</td>
<td>9984</td>
<td>9981</td>
<td>1337</td>
<td>3016.0</td>
</tr>
<tr>
<td>c30900</td>
<td>100143</td>
<td>100434</td>
<td>100427</td>
<td>5226</td>
<td>7185.0</td>
</tr>
<tr>
<td>e30900</td>
<td>9316</td>
<td>9328</td>
<td>9324</td>
<td>1488</td>
<td>3221.3</td>
</tr>
<tr>
<td>c60900</td>
<td>54551</td>
<td>54606</td>
<td>54544</td>
<td>982</td>
<td>9875.9</td>
</tr>
<tr>
<td>d60900</td>
<td>100103</td>
<td>100169</td>
<td>100146</td>
<td>6834</td>
<td>9466.9</td>
</tr>
</tbody>
</table>

Other interesting observation is that the resolution time required by each LP subproblem solved by the column generation method exceeds by far the resolution time of each knapsack subproblem as the instance size and the number of iteration increases. This explains the fact that, in the instances c10400, c20400, c30900, c30900 and c60900, the resolution time of our method is smaller than that of Pigatti et al. (2005) although the number of iterations is larger. In principle, this is surprising since the knapsack solver used by Pigatti et al. (2005) is several times faster than the standard dynamic programming required by our method. Finally, we observe that the approximate lower bounds achieved by our method are very close to the optimal ones, and even optimal in 7 out of 16 instances where the optimal lower bound is known (considering that their values can be rounded up since the objective function is always integer). For the instances where the optimal lower bound was not achieved, the difference was only 1 unit.

7 Conclusions

In this paper, we proposed a new method for smoothing the lagrangean dual function when the lagrangean subproblem is solved via dynamic programming. As a result, we are able to find good approximations of the optimal lagrangean dual cost in reasonable times. We also prove that, when the method converges to a optimal solution of the smoothed dual function, an associated fractional feasible solution for the primal problem is also provided. In addition, we provided experiments where we apply our technique to the GAP and compared against a stabilized column generation procedure proposed in (Pigatti et al., 2005). The reported results show that our method is suitable for instances where the
convergence of the column generation is poor. This happens when the ration $m/n$ is small. Finally, we believe that the existence of very efficient implementations for solving knapsack subproblems was an important fact against our method in the reported experiments. Due to this fact, we believe that the technique proposed in this paper can potentially achieve much better results for other combinatorial optimization problems where the best way to solve the subproblems is through the standard dynamic programming and the dual problem is hard for column generation methods (Rodrigues et al., 2008; Pessoa et al., 2008).

Acknowledgements AAP was partially financed by CNPq grants 301175/2006-3.

References


