CAHIER DU LAMSADE

Laboratoire d'Analyse et Modélisation de Systèmes pour l'Aide à la Décision
(Université Paris-Dauphine)
Unité de Recherche Associée au CNRS ESA 7024

THE GREEDY APPROACH FOR THE CREW PAIRING PROBLEM: SOME CASES OF POLYNOMIAL SUBPROBLEMS

CAHIER N° 159
janvier 1999

Laurent ALFANDARI [1]

received: October 1998.

[1] *LAMSADE, Université Paris-Dauphine, Place du Maréchal De Lattre de Tassigny, 75775 Paris Cedex 16, France (alfanla@lamsade.dauphine.fr).*

CONTENTS

L'approche gloutonne pour le problème de construction de rotations: cas de polynomialité du sous-problème

Résumé

Nous reprenons l'algorithme glouton classique pour le Set Covering en l'appliquant au problème de construction de rotations et montrons que, dans certains cas, la résolution du sous-problème - consistant à sélectionner itérativement la rotation minimisant le ratio "coût / nombre de tâches non couvertes" - s'effectue polynomialement en le nombre de tâches par un algorithme de plus court chemin à étiquettes multiples dans un graphe. Nous discutons également des liens qui unissent l'approche gloutonne et l'approche génération de colonnes, cette dernière pouvant également faire appel à la résolution d'un sous-problème de plus court chemin.

Mots-clés : *construction de rotations, glouton, plus court chemin, génération de colonnes.*

The greedy approach for the Crew Pairing Problem: some cases of polynomial subproblems

Abstract

We apply the usual greedy algorithm for the set covering to the Crew Pairing Problem and show that in some cases the subproblem - which consists of selecting the pairing minimizing a ratio "cost / number of uncovered tasks"- is solvable in polynomial time by a multi-label shortest path algorithm in a graph. We also discuss the links between the greedy approach and the column generation approach, as this latter one also often requires the solving of a shortest path subproblem.

Keywords : *crew pairing, set covering, greedy, shortest path, column generation.*

1 The Crew Pairing Problem

Given a set of tasks $V = \{1, 2, \ldots, n\}$ to be covered by crews (flights by stewards and hostesses, bus lines by drivers,...) [1], and given, for every task $j \in V$,
- a starting time $t_S(j)$ and an arrival time $t_A(j)$,
- a starting place $p_S(j)$ and an arrival place $p_A(j)$,
given a place B called "base", one constructs an acyclic oriented graph $G = (V \cup \{b\} \cup \{b'\}, E)$ with a cost function on arcs $d : E \to \mathrm{N}$, and $(i, j) \in E$ iff one of the following three conditions is satisfied:

$$\begin{cases} (i,j) \in V^2 \text{ and task } j \text{ can be performed after task } i, \\ i = b \text{ et } p_S(j) = B, \\ j = b' \text{ et } p_A(i) = B. \end{cases}$$

A crew pairing is a feasible sequence of tasks of V, starting and arriving in B, i.e, a path from source b to sink b' in graph G. The Crew Pairing Problem (CPP) consists then of finding a subset of pairings which covers V and minimizes the sum of the costs of the pairing. Let $V(p)$, $E(p)$ and $c(p)$ denote the set of tasks, the set of arcs and the cost of a path p, respectively; we formally define the problem in the following way.

Definition 1. Let $\mathcal{C}$ denote the set of paths from b to b' in graph G, and let $\mathcal{C}_A \subseteq \mathcal{C}$ denote the subset of paths from b to b' that satisfy some set of constraints A. The problem CPP(A) consists of finding in G a set of paths $\mathcal{C}' \subset \mathcal{C}_A$ such that $\cup_{p \in \mathcal{C}'} V(p) = V$ and the sum of the costs of the paths $\sum_{p \in \mathcal{C}'} c(p)$ is minimum.∎

The possibility of connecting two tasks i and j is obviously constrained by the chronological order of the tasks ($t_S(j) > t_A(i)$), and by the social regulation of the company (minimum rest time between two tasks, minimum lunch period, etc.). Let us remark that, for some sets of constraints A, connections (i, j) and (j, k) do not induce the feasibility of the subpath (i, j, k), and even finding a feasible path can be NP-hard (resource constraints for instance). For the rest of the paper, we assume that the cost of a path is an additive function of the valuations on the arcs.

Definition 2. The cost of a path $p \in \mathcal{C}_A$ is $c(p) = \sum_{e \in E(p)} d(e)$.∎

Let $\Gamma(i)$ (resp., $\Gamma^{-1}(i)$) denote the set of successors (resp., predecessors) of vertex i in G, and consider a path $p = (b, i_1, i_2, \ldots, i_t, b')$ of $\mathcal{C}_A$. Define the Total Rest Time of path p $TRT(p) = \sum_{1 \le l \le t-1} t_S(i_{l+1}) - t_A(i_l)$, and the Total Absence Time $TAT(p) = t_A(i_t) - t_S(i_1)$. If the total rest time is to be minimized, i.e, $c(p) = TPT(p)$, we set $d(i,j) = t_S(j) - t_A(i)$, $(i,j) \in V^2$, $(i,j) \in E$, $d(b,j) = 0$ $\forall j \in \Gamma(b)$, $d(i,b') = 0$ $\forall i \in \Gamma^{-1}(b')$. Generally speaking, if the cost of a path is a linear function of TAT and TRT, i.e, $\exists(\alpha, \beta, \gamma) \in \mathbb{R}^3$, $c(p) = \alpha TAT(p) + \beta TPT(p) + \gamma$, we can set $d(i,j) = \alpha(t_A(j) - t_A(i)) + \beta(t_A(j) - t_S(i))$ for $(i,j) \in V^2$, $(i,j) \in E$, $d(b,j) = \beta(t_A(j) - t_S(j)) + \gamma$ for $j \in \Gamma(b)$, and $d(i,b') = 0$ for $i \in \Gamma^{-1}(b')$.

Now we briefly present the more common approach to solve CPP, the "column generation" approach, as this technique presents interesting similarities with the greedy approach we develop further.

[1] These tasks may be composed of sub-tasks, cf. [9]

2 Linear programming and column generation

For $p \in \mathcal{C}_A$, we define variable $x_p = 1$ if path p is chosen in the solution, 0 otherwise, and parameter vector C_p, called "column", with components $C_{p,i} = 1$ if vertex $i \in V$ belongs to path p, 0 otherwise, $i = 1, \ldots, n$. The Crew Pairing Problem (CPP) is defined by the following integer linear program:

$$\begin{aligned} & min \sum_{p \in \mathcal{C}_A} c_p x_p \\ s.c.\ \forall i = 1, \ldots, n & \sum_{p \in \mathcal{C}_A} C_{p,i} x_p \geq 1 \qquad (P) \\ & x_p \in \{0; 1\},\ p \in \mathcal{C}_A \end{aligned}$$

The continuous relaxation of problem P will be noted $\bar{P}$. Let us remark that exhaustive explicitation of $\mathcal{C}_A$ is impossible for large-scale problems, as the total number of paths $|\mathcal{C}_A|$ usually grows exponentially in n. The column generation approach can get round this obstacle by solving $\bar{P}$, at each step t, on a restricted subset of columns $\mathcal{C}_t \subset \mathcal{C}_A$, by an adaptation of the simplex algorithm. A convincing example of this technique can be found in [9]. The key idea is the following. Given a feasible solution $\mathcal{C}_0 \subset \mathcal{C}_A$ for P, one solves the continuous program $\bar{P}_0$, which is the restriction of $\bar{P}$ to the set of columns $\mathcal{C}_0$:

$$\begin{aligned} & min \sum_{p \in \mathcal{C}_0} c_p x_p \\ s.c.\ \forall i = 1, \ldots, n & \sum_{p \in \mathcal{C}_0} C_{p,i} x_p \geq 1 \qquad (\bar{P}_0) \\ & x_p \in [0; 1],\ p \in \mathcal{C}_0 \end{aligned}$$

Solving $\bar{P}_0$ produces a vector of simplex multipliers $(\pi_1^0, \pi_2^0, \ldots, \pi_n^0)$ associated to tasks $1, 2, \ldots, n$. One looks then for the column with smallest negative reduced cost among all columns of $\mathcal{C}_A$ (so that the optimality of the procedure is guaranteed), i.e, column

$$C_{p_0^*} = arg \min_{C_p \in \mathcal{C}_A} c_p - \sum_{i=1}^{n} \pi_i^0 C_{p,i}$$

For some sets of contraints A, the column with smallest negative reduced cost $C_{p_0^*}$ can be found in time polynomial in n, without expliciting the whole set $\mathcal{C}_A$ of columns, by a shortest path algorithm in G [2], where a new cost $d^0(i,j) = d(i,j) - \pi_j^0$ is associated to arc $(i,j) \in E$. The column $C_{p_0^*}$ is added then to the set $\mathcal{C}_0$ - or even better, all columns with negative reduced cost are added -, forming a new set of columns $\mathcal{C}_1$, and the relaxed program is solved again on this new set of columns. At each step t, one solves then the following restricted program $\bar{P}_t$:

$$\begin{aligned} & min \sum_{p \in \mathcal{C}_t} c_p x_p \\ s.c.\ \forall i = 1, \ldots, n & \sum_{p \in \mathcal{C}_t} C_{p,i} x_p \geq 1 \qquad (\bar{P}_t) \\ & x_p \in [0; 1],\ p \in \mathcal{C}_t \end{aligned}$$

where $\mathcal{C}_t = \mathcal{C}_{t-1} \cup \{C_p \in \mathcal{C}_A :\ c_p - \sum_{i=1}^{n} \pi_i^{t-1} C_{p,i} < 0\}$ [3].

[2] rest time constraints in [9] require the computation of shortest paths in an expanded graph G', but the algorithm remains the same

[3] where π_i^t denotes the simplex multiplier associated to task i at step t

The columns added at step t correspond to the paths with negative reduced cost computed by a shortest path algorithm in graph G with valuations on arcs $d^t(i,j) = d(i,j) - \pi_j^t$. The continuous optimum is obtained as soon as, $\forall p \in \mathcal{C}_A,\ c_p - \sum_{i=1}^{n} \pi_i^t C_{p,i} \geq 0$ (all reduced costs are non negative), i.e as soon as the label of sink b' is positive in the shortest path computation.

Let us remark that the above procedure is nothing but the simplex algorithm, with the particularity that the column with minimum reduced cost is not found by exhaustive enumeration of the set of columns, but by solving a polynomial optimization subproblem (shortest path). By the way, this method solves the continuous version of the problem; in case of a non-integer solution, finding an optimal integer solution from the continuous solution may require much extra time computation. We now use the greedy master-slave technique in order to construct a feasible integer solution with performance guarantee. This greedy approach shares with the column generation approach the nice property that in both cases, the "master" problem calls a "slave" subproblem, polynomial in n for some sets of constraints, by implicitely exploring the set of feasible paths.

3 The greedy approach

Consider the following greedy heuristic (`Greedy` in short) for the problem CPP. At each step, find a path minimizing the ratio "cost of the path / number of uncovered tasks of the path", add this path to the solution and reiterate until all tasks of V are covered. We set, for $i \in V$, $u(i) = 1$ if vertex (task) i is covered at current step of `Greedy`, $u(i) = 0$ otherwise, and we define the following subproblem for CPP.

Definition 3. Given $u : V \to \{0;1\}$ identifying if vertex v is covered or not at some step of `Greedy`, the subproblem of finding a path p^* minimizing the ratio $c(p)\ /\ \sum_{i \in V(p)} u(i)$ over all paths p of $\mathcal{C}_A$ is called the "slave" problem. ∎

The greedy algorithm for the master problem is then the following:

```
ALGORITHM /Greedy/
    C' ← ∅;
    for i ∈ V do u(i) ← 1;
    while ∃i ∈ V : u(i) = 1 do
          find path p* = argmin_{p∈C_A} { c(p) / Σ_{i∈V(p)} u(i) };
          C' ← C' ∪ {p*};
          for i ∈ V(p*) do u(i) ← 0;
    end
    output C';
END OF /Greedy/
```

This algorithm provides a performance bound given in the following proposition.

Proposition 1. *For an instance I of CPP, let* `Greedy`(I) *and $OPT(I)$ denote the cost of the solution given by* `Greedy` *and the cost of an optimal solution, respectively. Then, for all I,*

$$\texttt{Greedy}(I) \leq (1 + \ln s) OPT(I)$$

where s is the maximum number of tasks of a path, i.e, $s = \max_{p \in \mathcal{C}_A} |V(p)|$.

This bound is derived from the analysis of the classical greedy algorithm for the Set Covering Problem by Chvátal ([5]). Greedy is, to our knowledge, the only heuristic for CPP with worst-case performance guarantee. Baker implemented Greedy for some Crew Pairing problems applied to air transportation([4]); his results are reported in the following table.

n	$\|\mathcal{C}_A\|$	OPT	Greedy	%
36	506	4 424	4 424	0
180	4534	27 803	28 564	2.7
200	3208	30 110	32 095	6.5
335	5290	48 089	51 344	6.7
376	1245	74 679	74 679	0
426	2780	35 974	36 550	1.6
180	2275	28 759	29 876	3.8
180	3431	27 936	33 217	18.8
376	4802	72 935	81 979	12.4

The computational results above look convincing enough to show the practical efficiency of Greedy. Our contribution is more of theoretical order : whereas the set of feasible paths $\mathcal{C}_A$ was a data in [4], and the complexity of Greedy was polynomial in $|\mathcal{C}_A|$, i.e in $O(2^n)$ at worst case (the optimal path p^* of definition 3 was found by exhaustive enumeration of $\mathcal{C}_A$), we exhibit cases where the set of feasible paths $\mathcal{C}_A$ is not known a priori, and where Greedy is computed in time polynomial in n, the number of tasks, by solving the slave in time polynomial in n.

3.1 Case without constraints

Let us assume that every path from b to b' is feasible in graph G. We consider then the set of paths $\mathcal{C}_{A_1} = \mathcal{C}$. This case is not a trivial case: the rest time constraints in [9] refer to the present case via transforming the initial graph. Now we prove that the slave problem is solvable in time polynomial in n by showing that, graph G being acyclic, the following extension of Bellman's sub-optimality principle holds for CPP(A_1): *if $(b, i_1, i_2, \ldots, i_t, j)$ is a shortest path among the paths from b to j containing exactly k uncovered vertices, then $(b, i_1, i_2, \ldots, i_t)$ is a shortest path among the paths from b to i_t containing exactly $k - u(j)$ uncovered vertices.*
Consider the following variant of the classical shortest path algorithm for acyclic graphs (cf. [7], p.49), where label $\lambda^k(j)$ is the value of a shortest path among paths from b to j containing exactly k uncovered vertices of V, $k = 0, \ldots, s$, with $s = \max_{p \in \mathcal{C}_A} |V(p)|$ [4].

ALGORITHM /S1(u)/
$S \leftarrow \{b\}$;
$\lambda^0(b) \leftarrow 0$; $\lambda^k(b) \leftarrow \infty$, $k = 1, \ldots, s$;
while there exists a vertex $j \notin S$ such that $\Gamma^{-1}(j) \subset S$ **do**
$S \leftarrow S \cup \{j\}$;
compute, for $k = 1, \ldots, s$, label $\lambda^k(j) \leftarrow \min_{i \in \Gamma^{-1}(j)} \{\lambda^{k-u(j)}(i) + d(i,j)\}$;
compute $\lambda^0(j) \leftarrow \min_{i \in \Gamma^{-1}(j)} \{\lambda^0(i) + d(i,j)\}$ if $u(j) = 0$, else $\lambda^0(j) \leftarrow \infty$;
end
output path p* minimizing $\lambda^k(b')/k$ for $k = 1, \ldots, s$;
END OF /S1(u)/

[4] the value s can be computed by a longuest path algorithm in G

Theorem 1. *Algorithm* S1 *solves in* $O(n^3)$ *the slave problem associated to CPP*(A_1).

Proof: The complexity of algorithm S1 is in $O(ns\Delta)$, where $\Delta = \max_{j \in V \cup \{b'\}} |\Gamma^{-1}(j)|$, i.e in $O(n^3)$. It remains to prove that the algorithm is optimal. As the cost of p^* is $\min_k \lambda^k(b')/k$, optimality is proved if, for $j \in V \cup \{b'\}$, label $\lambda^k(j)$ is indeed the value of a shortest path among paths from b to j containing exactly k uncovered vertices. We shall first show that $\lambda^k(j)$ is a lower bound of this value, then we show this bound is achieved. Let $p = (b, i_1, i_2, \ldots, i_t, j)$ be a path from b to j in G satisfying $\sum_{1 \leq l \leq t} u(i_l) + u(j) = k$. As $\forall j, \forall i \in \Gamma^{-1}(j)$, $\forall k$, $\lambda^k(j) \leq \lambda^{k-u(j)}(i) + d(i,j)$, it follows that

$$\lambda^k(j) \leq \lambda^{k-u(j)}(i_t) + d(i_t, j)$$
$$\lambda^{k-u(j)}(i_t) \leq \lambda^{k-u(j)-u(i_t)}(i_{t-1}) + d(i_{t-1}, i_t)$$
$$\ldots$$
$$\lambda^{k-\ldots-u(i_3)}(i_2) \leq \lambda^{k-\ldots-u(i_2)}(i_1) + d(i_1, i_2)$$
$$\lambda^{k-\ldots-u(i_2)}(i_1) \leq \lambda^{k-\ldots-u(i_1)}(b) + d(b, i_1)$$

Summing these $t+1$ inequalities term by term we get

$$\lambda^k(j) \leq \lambda^{k-u(j)-\sum_{1 \leq l \leq t} u(i_l)}(b) + d(b, i_1) + d(i_1, i_2) + \ldots + d(i_{t-1}, i_t) + d(i_t, j)$$

Since $u(j) + \sum_{1 \leq l \leq t} u(i_l) = k$ and $\lambda^0(b) = 0$, we obtain for every path p from b to j,

$$\lambda^k(j) \leq c(p)$$

The value $\lambda^k(j)$ is then a lower bound of the value of a shortest path among paths from b to j containing exactly k uncovered vertices. Moreover, this bound is attained; we construct a path p^* from b to j of value $\lambda^k(j)$ the following way, classical when dealing with shortest path problems: start from j and, going back to the root b, connect j to the predecessor i of j such that $\lambda^{k-u(j)}(i) + d(i,j)$ is minimum, then connect i to the predecessor h of i such that $\lambda^{k-u(j)-u(i)} + d(h,i)$ is minimum, and reiterate until b is attained. ∎

Let us consider now the case when all paths from b to b' are not feasible.

3.2 Case of resource constraints

Assume that a pairing p is feasible if it contains at most one lunch period and one night period, and if the total number of effective worked hours $TAT(p) - TPT(p)$ is at most 30 hours. Typically, given q resources, a path is feasible if, for $h = 1, \ldots, q$, the cumulative amount of resource h on this path is lower than a given threshold m_h. Let $r_h(e) \in \mathbb{N}$ denote the consumption of resource h on arc e, $\mathbf{m} = (m_1, \ldots, m_q)$, and define the following problem.

Definition 4. The problem CPP(A_2) is the restriction of CPP to the set of paths $\mathcal{C}_{A_2} = \{p \in \mathcal{C} : \exists r_1 : E \to \mathbb{N}, \ldots, \exists r_q : E \to \mathbb{N}, \exists \mathbf{m} \in \mathbb{N}^q, \sum_{e \in E(p)} r_h(e) \leq m_h, h = 1, \ldots, q\}$. ∎

In the restricted case where all values m_h are low, we propose an exact algorithm solving the slave problem, based on the following extension of Bellman's sub-optimality principle for CPP(A_2): *if* $(b, i_1, i_2, \ldots, i_t, j)$ *is a shortest path among paths from* b *to* j *containing* k *uncovered vertices and for which the cumulative amount of resource* h *is* x_h *for* $h = 1, \ldots, q$, *then* $(b, i_1, i_2, \ldots, i_t)$ *is a shortest path among paths from* b *to* j *containing* $k - u(j)$ *uncovered vertices and for which the cumulative amount of each resource* h *is* $x_h - r_h(i_t, j)$.

Let us note for $(i,j) \in E$, $\mathbf{r}_{ij} = (r_1(i,j), r_2(i,j), \ldots, r_q(i,j))$, and for $\mathbf{x}, \mathbf{y} \in \mathbb{N}^q$, $\mathbf{x} \leq \mathbf{y}$ iff $x_h \leq y_h$, $\forall h = 1, \ldots, q$, and $\mathbf{x} - \mathbf{y} = (x_1 - y_1, \ldots, x_q - y_q)$. Consider now the following algorithm solving the slave problem, where label $\lambda^k_{\mathbf{x}}(j)$ is the value of a shortest path among paths p from b to j satisfying $\sum_{v \in V(p)} u(v) = k$ and $\sum_{e \in E(p)} r_h(e) = x_h$, $h = 1, \ldots, q$.

```
ALGORITHM /S2(u)/
    S ← {b};
    λ⁰_0(b) ← 0;
    λ^k_x(b) ← ∞, k = 0, ..., s, x ≤ m, (k, x) ≠ (0, 0);
    while there exists a vertex j ∉ S such that Γ⁻¹(j) ⊂ S do
        S ← S ∪ {j};
        for x ≤ m do compute labels
            λ^k_x(j) ← min_{i∈Γ⁻¹(j), x≥r_ij} { λ^{k-u(j)}_{x-r_ij}(i) + d(i,j) }[5], k = 1, ..., s;
            λ⁰_x(j) ← min_{i∈Γ⁻¹(j), x≥r_ij} { λ⁰_{x-r_ij}(i) + d(i,j) } if u(j) = 0 else λ⁰_x(j) ← ∞;
        end
    end
    output path p* minimizing λ^k_x(b')/k for k = 1, ..., s, x ≤ m;
END OF /S2(u)/
```

[5] *if* $\forall i \in \Gamma^{-1}(j)$, $\neg(\mathbf{x} \geq \mathbf{r}_{ij})$, *then* $\lambda^k_{\mathbf{x}}(j) \leftarrow \infty$.

Theorem 2. *Algorithm* S2 *solves the slave problem for CPP(A_2) in* $O(n^3 \prod_{h=1}^{q} (m_h + 1))$.

Proof: The complexity of S2 is in $O(ns\Delta \prod_{1 \leq h \leq q}(m_h + 1))$, i.e, in $O(n^3 \prod_{1 \leq h \leq q}(m_h + 1))$. Now, the slave problem is solved if $\lambda^k_{\mathbf{x}}(j)$ is indeed the value of a shortest path among paths p from b to j satisfying $\sum_{i \in V(p)} u(i) = k$ and $\sum_{e \in E(p)} r_h(e) = x_h$, $h = 1, \ldots, q$ (then a path minimizing $\lambda^k_{\mathbf{x}}(b')/k$ for $k = 1, \ldots, s$, $\mathbf{x} \leq \mathbf{m}$, is indeed an optimal path). Let $p = (b, i_1, i_2, \ldots, i_t, j)$ be a path satisfying the above conditions. As $\forall j, \forall i \in \Gamma^{-1}(j), \forall k, \lambda^k_{\mathbf{x}}(j) \leq \lambda^{k-u(j)}_{\mathbf{x}-\mathbf{r}_{ij}}(i) + d(i,j)$, we get

$$\lambda^k_{\mathbf{x}}(j) \leq \lambda^{k-u(j)}_{\mathbf{x}-\mathbf{r}_{i_t j}}(i_t) + d(i_t, j)$$

$$\lambda^{k-u(j)}_{\mathbf{x}-\mathbf{r}_{i_t j}}(i_t) \leq \lambda^{k-u(j)-u(i_t)}_{\mathbf{x}-\mathbf{r}_{i_t j}-\mathbf{r}_{i_{t-1} i_t}}(i_{t-1}) + d(i_{t-1}, i_t)$$

$$\ldots$$

$$\lambda^{k-\ldots-u(i_3)}_{\mathbf{x}-\ldots-\mathbf{r}_{i_2 i_3}}(i_2) \leq \lambda^{k-\ldots-u(i_2)}_{\mathbf{x}-\ldots-\mathbf{r}_{i_1 i_2}}(i_1) + d(i_1, i_2)$$

$$\lambda^{k-\ldots-u(i_2)}_{\mathbf{x}-\ldots-\mathbf{r}_{i_1 i_2}}(i_1) \leq \lambda^{k-\ldots-u(i_1)}_{\mathbf{x}-\ldots-\mathbf{r}_{b i_1}}(b) + d(b, i_1)$$

By summing these $t+1$ inequalities we obtain

$$\lambda^k_{\mathbf{x}}(j) \leq \lambda^{k-u(j)-\ldots-u(i_1)}_{\mathbf{x}-\mathbf{r}_{i_t j}-\ldots-\mathbf{r}_{b i_1}}(b) + d(b, i_1) + d(i_1, i_2) + \ldots + d(i_t, j)$$

Since $u(i_1) + \ldots + u(i_t) + u(j) = k$, $\mathbf{r}_{bi_1} + \mathbf{r}_{i_1 i_2} + \ldots + \mathbf{r}_{i_t b'} = \mathbf{x}$, and $\lambda^0_{\mathbf{0}}(b) = 0$, the following inequality finally holds:

$$\lambda^k_{\mathbf{x}}(j) \leq c(p)$$

Hence we showed that $\lambda^k_{\mathbf{x}}(j)$ is a lower bound of the value of the shortest path mentioned above, and this bound is attained by constructing the path as previously.∎

The complexity of S2 is polynomial if all values m_h are polynomially bounded, the algorithm running in $O(n^3)$ if these values are fixed constants independent of n. The applicability of algorithm S2 for resource constraints is thus restricted to "easy" resources (such as the number of stops, breaks, or nights spent out of the base). The case of time resources is more intricate. For instance, assume that the total effective worked time of a pairing is constrained by an upper bound, i.e, $\exists m \in \mathbb{N},\ p \in \mathcal{C}_A \Longrightarrow TAT(p) - TPT(p) \leq m$; as the minute is often chosen as time unit, m can be quite a high value for weekly pairings. On the opposite, the constraint $TAT(p) \leq m$ is not an obstacle as $TAT(p)$ only depends of the first task and the last task of the pairing (see [9]). Generally speaking, the Shortest Path Problem with resource constraints is NP-hard ([6], ND30).

Finally, let us remark that algorithm S2 can be easily modified in order to solve, in a column generation approach for CPP(A_2), the subproblem which consists of finding at step t the path with smallest reduced cost; simply compute label $\lambda_{\mathbf{x}}(j) \leftarrow \min_{i \in \Gamma^{-1}(j), \mathbf{x} \geq \mathbf{r}_{ij}} \{\lambda_{\mathbf{x}-\mathbf{r}_{ij}}(i) + d(i,j) - \pi_j^t\}$ in the while loop, and finally output path of value $\min_{\mathbf{x} \leq \mathbf{m}} \lambda_{\mathbf{x}}(b')$.

4 Conclusion

The classical greedy algorithm for the set covering problem has showed great efficiency in crew pairing contexts([4]), as well as interesting complementarity with the column generation approach, since the greedy provides a good integer solution that can initialize a simplex computation. We also showed the similarity of the two approaches by exhibiting some simple sets of constraints that make the subproblem solvable in polynomial time, in both cases, by a shortest path algorithm in a graph, avoiding exhaustive explicitation of the feasible pairings. Furthermore, we conjecture that, given a set of constraints A on the feasible paths, the subproblem in the column generation approach is solvable in polynomial time if and only if the slave problem in the greedy approach is solvable in polynomial time for CPP(A). This greedy technique can be applied in fact to a larger class of covering and partitioning problems called "master-slave tractable" problems, by exploiting an unusual reduction to the set covering problem (see [2], [10] for more details).

References

[1] Alfandari L. and Paschos V. Th. 1997. *Master-slave strategies and polynomial approximation*, manuscript.

[2] Alfandari L. and Paschos V. Th. 1998. *On the approximation of some spanning arborescence problems*, Proceed. Int. Symp. on Comp. and Inf. Sc. ISCIS 98, 293-301.

[3] Alfandari L. and Paschos V. Th. 1998. *Approximating minimum spanning tree of depth 2*, manuscript.

[4] Baker E. 1981. *Efficient heuristic algorithms for the weighted set covering problem*, Comput. & Op. Res. vol.8, No 4, 303-310.

[5] Chvátal V. 1979. *A greedy-heuristic for the set covering problem*, Math. Oper. Res. 4, 233-235.

[6] Garey M. R. and Johnson D. S. 1979. *Computers and intractability. A guide to the theory of NP-completeness*, W. H. Freeman, San Francisco.

[7] Gondran M. and Minoux M. 1979. *Graphes et algorithmes*, Eyrolles, Paris.

[8] Johnson D. S. 1974. *Approximation algorithms for combinatorial problems*, J. Comput. System Sci. 9, 256-278.

[9] Lavoie S., Minoux M. and Odier E. 1988. *A new approach for crew pairing problems by column generation with an application to air transportation*, EJOR 35, 45-58.

[10] Simon H. U. 1990. *On approximate solutions for combinatorial optimization problems*, SIAM J. Disc. Math. 3(2), 294-310.

[11] Slavík P. 1996. *A tight analysis of the greedy algorithm for set cover*, ACM-STOC 435-441.