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

CAHIER N° 159
janvier 1999

Laurent ALFANDARI

received: October 1998.

1 LAMSADÉ, Université Paris-Dauphine, Place du Maréchal De Lattre de Tassigny, 75775 Paris Cedex 16, France (alfonla@lamsade.dauphine.fr).
## CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Pages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Résumé</td>
<td>i</td>
</tr>
<tr>
<td>Abstract</td>
<td>i</td>
</tr>
<tr>
<td>1. The crew pairing problem</td>
<td>1</td>
</tr>
<tr>
<td>2. Linear programming and column generation</td>
<td>2</td>
</tr>
<tr>
<td>3. The greedy approach</td>
<td>3</td>
</tr>
<tr>
<td>3.1 Case without constraints</td>
<td>4</td>
</tr>
<tr>
<td>3.2 Case of resource constraints</td>
<td>5</td>
</tr>
<tr>
<td>4. Conclusion</td>
<td>7</td>
</tr>
<tr>
<td>References</td>
<td>8</td>
</tr>
</tbody>
</table>
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 polynomiallement 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 short-
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 $d : E \rightarrow \mathbb{N}$, and $(i, j) \in E$ iff one of the following three conditions is satisfied:

\[
\begin{align*}
(i, j) &\in V^2 \text{ and task } j \text{ can be performed after task } i, \\
&\text{if} \quad i = b \quad \text{et} \quad p_S(j) = B, \\
&\text{if} \quad j = b' \quad \text{et} \quad p_A(i) = B.
\end{align*}
\]

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...
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:

\[
\min \sum_{p \in \mathcal{C}_A} c_p x_p
\]
\[\text{s.c. } \forall i = 1, \ldots, n \sum_{p \in \mathcal{C}_A} C_{p,i} x_p \geq 1 \quad (P)\]
\[x_p \in \{0, 1\}, \ p \in \mathcal{C}_A\]

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 \( \mathcal{C}_0 \):

\[
\min \sum_{p \in \mathcal{C}_0} c_p x_p
\]
\[\text{s.c. } \forall i = 1, \ldots, n \sum_{p \in \mathcal{C}_0} C_{p,i} x_p \geq 1 \quad (\bar{P}_0)\]
\[x_p \in \{0, 1\}, \ p \in \mathcal{C}_0\]

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

\[
C_{\mathcal{C}_0^*} = \arg \min_{C_p \in \mathcal{C}_A} c_p - \sum_{i=1}^{n} \pi_i^0 C_{p,i}
\]

For some sets of constraints \( A \), the column with smallest negative reduced cost \( C_{\mathcal{C}_0^*} \) can be found in time polynomial in \( n \), without expliciting the whole set \( \mathcal{C}_A \) of columns, by a shortest path algorithm in \( G \), where a new cost \( d^0(i, j) = d(i, j) - \pi_j^0 \) is associated to arc \((i, j) \in E \). The column \( C_{\mathcal{C}_0^*} \) 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 \( \bar{P}_t \):

\[
\min \sum_{p \in \mathcal{C}_t} c_p x_p
\]
\[\text{s.c. } \forall i = 1, \ldots, n \sum_{p \in \mathcal{C}_t} C_{p,i} x_p \geq 1 \quad (\bar{P}_t)\]
\[x_p \in \{0, 1\}, \ p \in \mathcal{C}_t\]

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

\text{[3]} rest time constraints in [9] require the computation of shortest paths in an expanded graph \( G' \), but the algorithm remains the same.

\text{[3]} where \( \pi_i^t \) 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 $d'(i,j) = d(i,j) - \pi_j^t$. The continuous optimum is obtained as soon as, $\forall p \in C_A$, $c_p - \sum_{i=1}^{n} \pi_i^t C_{p,i} \geq 0$ (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 implicitly 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.
This bound is derived from the analysis of the classical greedy algorithm for the Set Covering Problem by Chvátal ([5]). Greedy, 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.

<table>
<thead>
<tr>
<th>n</th>
<th></th>
<th>C_A</th>
<th></th>
<th>OPT</th>
<th>4 424</th>
<th>4 424</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>36</td>
<td>506</td>
<td>4 424</td>
<td>4 424</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>180</td>
<td>4534</td>
<td>27 803</td>
<td>28 564</td>
<td>2.7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>200</td>
<td>3208</td>
<td>30 110</td>
<td>32 095</td>
<td>6.5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>335</td>
<td>5290</td>
<td>48 089</td>
<td>51 344</td>
<td>6.7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>376</td>
<td>1245</td>
<td>74 679</td>
<td>74 679</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>426</td>
<td>2780</td>
<td>35 974</td>
<td>36 550</td>
<td>1.6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>180</td>
<td>2275</td>
<td>28 759</td>
<td>29 876</td>
<td>3.8</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>180</td>
<td>3431</td>
<td>27 936</td>
<td>33 217</td>
<td>18.8</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>376</td>
<td>4802</td>
<td>72 935</td>
<td>81 979</td>
<td>12.4</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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 C_A was a data in [4], and the complexity of Greedy was polynomial in |C_A|, i.e. in O(2^n) at worst case (the optimal path p^* of definition 3 was found by exhaustive enumeration of C_A), we exhibit cases where the set of feasible paths 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 v is feasible in graph G. We consider then the set of paths C_A = 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_i): if (b, i_1, i_2, ..., i_k, j) is a shortest path among the paths from b to j containing exactly k uncovered vertices, then (b, i_1, i_2, ..., i_k) is a shortest path among the paths from b to i_k containing exactly k - u(j) uncovered vertices.

Consider the following variant of the classical shortest path algorithm for acyclic graphs (cf. [7], [8]). Let V^*(b, j) be the value of a shortest path among paths from b to j containing exactly
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 $O(n(s\Delta))$, 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 $\sum_{1 \leq t \leq t} u(i_t) + u(j) = k$. As $\forall j, i \in V \cup \{b\}, \forall k, \lambda^k(j) \leq \lambda^{k-u(i)}(i) + d(i, j)$, it follows that

\[
\lambda^k(j) \leq \lambda^{k-u(i)}(i_t) + d(i_t, j)
\]

\[
\lambda^{k-u(i)}(i_t) \leq \lambda^{k-u(i)-u(i_t)}(i_{t-1}) + d(i_{t-1}, i_t)
\]

\[
\cdots
\]

\[
\lambda^{k-\cdots-u(i_2)}(i_2) \leq \lambda^{k-\cdots-u(i_2)}(i_1) + d(i_1, i_2)
\]

\[
\lambda^{k-\cdots-u(i_2)}(i_1) \leq \lambda^{k-\cdots-u(i_1)}(b) + d(b, i_1)
\]

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

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

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

**Algorithm /S2(\(u\))/**

\[
S \leftarrow \{b\}; \\
\lambda^0_{0}(b) \leftarrow 0; \\
\lambda^k_{x}(b) \leftarrow \infty, k = 0, \ldots, s, x \leq m, (k, x) \neq (0, 0); \\
\textbf{while} \text{ there exists a vertex } j \in S \text{ such that } \Gamma^{-1}(j) \subset S \text{ do} \\
\text{ for } x \leq m \text{ do compute labels} \\
\lambda^k_{x}(j) \leftarrow \min_{i \in \Gamma^{-1}(j), x \geq r_{ij}} \{ \lambda^{k-u(j)}_{x-r_{ij}}(i) + d(i, j) \}^5, k = 1, \ldots, s; \\
\lambda^k_{x}(j) \leftarrow \min_{i \in \Gamma^{-1}(j), x \geq r_{ij}} \{ \lambda^{k-u(j)}_{x-r_{ij}}(i) + d(i, j) \} \text{ if } u(j) = 0 \text{ else } \lambda^k_{x}(j) \leftarrow \infty; \\
\text{ end} \\
\text{end} \\
\text{ output path } p^* \text{ minimizing } \lambda^k_{x}(b')/k \text{ for } k = 1, \ldots, s, x \leq m; \\
\text{END OF /S2(\(u\))/} \\
^5 \text{ if } \forall i \in \Gamma^{-1}(j), -(x \geq r_{ij}), \text{ then } \lambda^k_{x}(j) \leftarrow \infty.
\]

**Theorem 2.** Algorithm S2 solves the slave problem for CPP(A2) in \(O(n^3 \prod_{h=1}^{q} (m_h + 1))\).

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

\[
\lambda^k_{x}(j) \leq \lambda^{k-u(j)}_{x-r_{ij}}(i) + d(i, j) \\
\lambda^{k-u(j)}_{x-r_{ij}}(i) \leq \lambda^{k-u(i)}_{x-r_{ij}-r_{i_{t-1}}}(i_{t-1}) + d(i_{t-1}, i_t) \\
\cdots \\
\lambda^{k-u(i_2)}_{x-r_{i_{t+1}}}(i_2) \leq \lambda^{k-u(i_1)}_{x-r_{i_{t+1}}}(i_1) + d(i_{t}, i_{t}) \\
\lambda^{k-u(i_1)}_{x-r_{i_{t+1}}}(i_1) \leq \lambda^{k-u(i_1)}_{x-r_{i_{t+1}}}(i_1) + d(b, i_1)
\]

By summing these \(t + 1\) inequalities we obtain

\[
\lambda^k_{x}(j) \leq \lambda^{k-u(i_{t+1})}_{x-r_{i_{t+1}}}(i_1) + d(b, i_1) + d(i_1, i_{t}) + \ldots + d(i_{t}, j)
\]

Since \(u(i_1) + \ldots + u(i_t) + u(j) = k, r_{bi_1} + r_{i_1i_2} + \ldots + r_{i_{t}i_{t+1}} = x,\) and \(\lambda^0_{0}(b) = 0,\) the following inequality finally holds:

\[
\lambda^k_{x}(j) \leq c(p)
\]

Hence we showed that \(\lambda^k_{x}(j)\) 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 $n_{ah}$ 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., $\exists m \in \mathbb{N}, \ p \in C_A \implies TAT(p) - TPT(p) \leq m$; 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 $\lambda_x(j) \leftarrow \min_{i \in \Gamma^{-1}(j), x \geq r_{ij}} \{\lambda_x - r_{ij}(i) + d(i,j) - \pi^T_j\}$ in the while loop, and finally output path of value $\min_{x \leq m} \lambda_x(\theta^*)$.

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


