HAL open science

The Biobjective Assignment Problem

Anthony Przybylski, Xavier Gandibleux, Matthias Ehrgott

To cite this version:

Anthony Przybylski, Xavier Gandibleux, Matthias Ehrgott. The Biobjective Assignment Problem. 2006. hal-00023155

HAL Id: hal-00023155

https://hal.science/hal-00023155v1

Preprint submitted on 20 Apr 2006

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

The Biobjective Assignment Problem

Anthony Przybylski [1], Xavier Gandibleux[1], Matthias Ehrgott [2]

[1] LINA - Laboratoire d'Informatique de Nantes Atlantique
FRE CNRS 2729
Université de Nantes
2, rue de la Houssinière BP 92208
F-44322 Nantes Cedex 03 - FRANCE

[2] Department of Engineering Science
University of Auckland
Private Bag 92019, Auckland – New Zealand

— Multiobjective Optimization —

LABORATOIRE D'INFORMATIQUE DE NANTES ATLANTIQUE

RESEARCH REPORT

N° 05.07

December 2005

UNIVERSITÉ DE NANTES

ECOLE DES MINES DE NANTES

CENTRE NATIONAL DE LA RECHERCHE SCIENTIFIQUE

Anthony Przybylski, Xavier Gandibleux, Matthias Ehrgott

The Biobjective Assignment Problem

34 p.

Les rapports de recherche du Laboratoire d'Informatique de Nantes-Atlantique sont disponibles aux formats PostScript® et PDF® à l'URL :
http://www.sciences.univ-nantes.fr/lina/Vie/RR/rapports.html

Research reports from the Laboratoire d'Informatique de Nantes-Atlantique are available in PostScript® and PDF® formats at the URL:
http://www.sciences.univ-nantes.fr/lina/Vie/RR/rapports.html

© December 2005 by Anthony Przybylski, Xavier Gandibleux, Matthias Ehrgott

The Biobjective Assignment Problem

Anthony Przybylski, Xavier Gandibleux, Matthias Ehrgott

Anthony.Przybylski@univ-nantes.fr, Xavier.Gandibleux@univ-nantes.fr,
M.Ehrgott@auckland.ac.nz

Abstract

In this paper we present a synthesis of the two phase method for the biobjective assignment problem. The method, which is a general technique to solve multiobjective combinatorial optimization (MOCO) problems, has been introduced by Ulungu in 1993. However, no complete description of the method to find all efficient solutions of the biobjective assignment problem (that allows an independent implementation) has been published.
First, we present a complete description of a two phase method for the biobjective assignment problem, with an improved upper bound. Second, a combination of this method with a population based heuristic using path relinking is proposed to improve computational performance. Third, we propose a new technique for the second phase with a ranking approach.
All of the methods have been tested on instances of varying size and range of objective function coefficients. We discuss the obtained results and explain our observations based on the distribution of objective function values.

Additional Key Words and Phrases: Multiobjective combinatorial optimization, assignment problem, exact algorithm, population based heuristic, path-relinking, efficient solutions, k-best solution, ranking

1 Introduction

1.1 The Assignment Problem with Two Objectives

The single objective assignment problem (AP) is an integer programming problem that can be solved as a linear program due to total unimodularity of the constraint matrix. Efficient algorithms to solve it, e.g., the Hungarian method or the successive shortest paths method [17, 1] are well known.

In this paper we consider the assignment problem with two objectives (BAP). It can be formulated as follows:

$$\begin{array}{rcll} \min z^k(x) & = & \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}^k x_{ij} & k=1,2 \\ & & \sum_{i=1}^{n} x_{ij} = 1 & j=1,\dots,n \\ & & \sum_{j=1}^{n} x_{ij} = 1 & i=1,\dots,n \\ & & x_{ij} \in \{0,1\} & i,j=1,\dots,n, \end{array} \tag{BAP}$$

where all objective function coefficients c_{ij}^k are non-negative integers and $x = (x_{11}, \dots, x_{nn})$ is the matrix of decision variables. For simplicity, we shall write $C(x)$ for $\sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{ij}$.

Let X denote the set of feasible solutions of (BAP). We call $\mathbb{R}^{n^2}, X \subset \{0,1\}^{n^2} \subset \mathbb{R}^{n^2}$, decision space and $\mathbb{R}^2, Z = \{z(x) : x \in X\} \subset \mathbb{N}^2 \subset \mathbb{R}^2$, objective space. Z is also called the feasible set in objective space.

In multiobjective optimization there is in general no feasible solution which minimizes all objectives simultaneously.

Definition 1. *A feasible solution $x^* \in X$ is called* efficient *if there does not exist any other feasible solution $x \in X$ such that $z^k(x) \leq z^k(x^*)$, $k = 1, 2$, with at least one strict inequality. $z(x^*)$ is then called a* nondominated point. *The set of efficient solutions is denoted by X_E and the image of X_E in Z is called the* nondominated frontier Z_N. *If $x, x' \in X$ are such that $z^k(x) \leq z^k(x'), k = 1, 2$, and $z(x) \neq z(x')$ we say that x dominates x' ($z(x)$ dominates $z(x')$).*

The set of of efficient solutions is partitioned in two subsets as follow:

- *Supported* efficient solutions are optimal solutions of a weighted sum single objective problem

$$\min\left\{\lambda^1 z^1(x) + \lambda^2 z^2(x) : x \in X, \lambda^1 > 0, \lambda^2 > 0\right\}. \tag{BAP$_\lambda$}$$

  All supported nondominated points are located on the "lower-left boundary" of the convex hull of Z ($\operatorname{conv} Z$), i.e., they are nondominated points of $(\operatorname{conv} Z) + \mathbb{R}^2_+$. We use the notations X_{SE} and Z_{SN}, respectively, to denote supported efficient solutions and supported nondominated points.

- *Nonsupported* efficient solutions are efficient solutions that are not optimal solutions of (BAP)$_\lambda$ for any λ with $\lambda^1, \lambda^2 > 0$. Nonsupported nondominated points are located in the interior of the convex hull of Z. No theoretical characterisation which leads to an efficient computation of the nonsupported efficient solutions is known. The sets of nonsupported efficient solutions and nondominated points are denoted X_{NE} and Z_{NN}, respectively.

In addition we can distinguish two classes of supported efficient solutions.

- Supported efficient solutions x are those whose objective vectors $z(x)$ are located on the vertex set of $\operatorname{conv} Z$. We call these extremal supported efficient solutions X_{SE1} and let $Z_{SN1} = z(X_{SE1})$, the extremal supported nondominated points.

- The set of those $x \in X_{SE}$ for which $z(x)$ is not located on the vertex set of $\operatorname{conv} Z$. This set is denoted by X_{SE2} and its image by Z_{SN2}.

Both X_{SE1} and X_{SE2} can be obtained by solving (BAP$_\lambda$). Neverthless, the computation of the latter is generally more difficult than that of the former, because for all efficient solutions x with $z(x)$ on an edge of the convex hull of Z, the weight vector λ for which x minimizes (BAP$_\lambda$) is the same. Therefore finding X_{SE2} requires an enumeration of *all* optimal solutions of (BAP$_\lambda$) (cf. Figure 1).

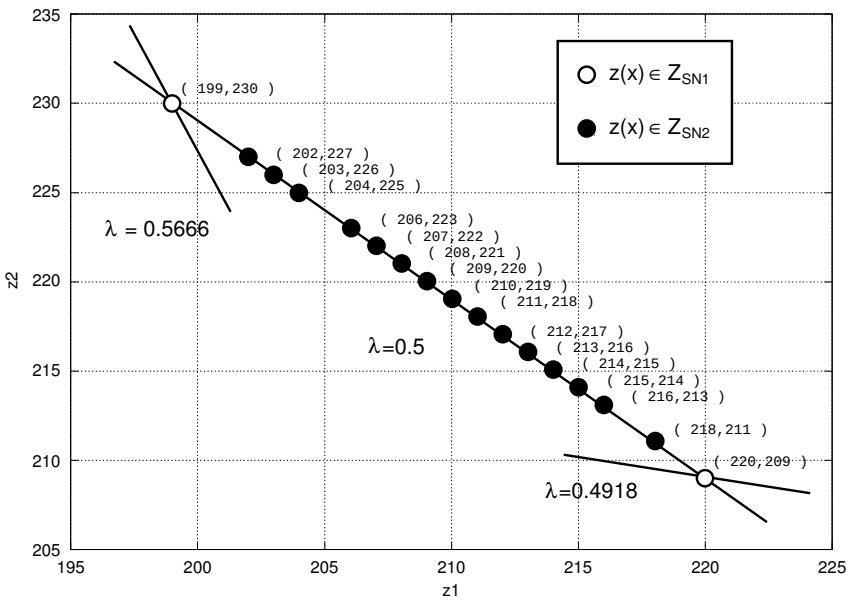

Figure 1: X_{SE2}: Supported efficient solutions with $z(x)$ on an edge of the convex hull of Z (instance `2AP100-1A20`).

All efficient solutions of multiobjective linear programming problems are supported (see, e.g., [19]). Because of total unimodularity of the constraint matrix of (BAP), one might therefore think that nonsupported efficient solutions of the (BAP) do not exist. But this is not true, as [22] have pointed out and as can be seen, e.g., in Figure 3. Thus, the biobjective assignment problem must be considered as an integer programming problem. Moreover, the following theorem holds.

Theorem 1. *The assignment problem with two objectives is $\mathcal{NP}$-complete, #$\mathcal{P}$-complete, and intractable.*

Here, intractability means that an exponential number of efficient solutions may exist. For a proof see [18] for $\mathcal{NP}$-completeness and [23] for #$\mathcal{P}$-completeness and intractability or [4] for a concise proof of all statements.

1.2 Classification of Efficient Solutions

X_E can be classified into subsets in several ways.

Definition 2. 1. *[13] Feasible solutions $x, x' \in X$ are said to be* equivalent *if $z(x) = z(x')$.*

2. *[13] A* complete set *X_E is a set of efficient solutions such that all $x \in X \setminus X_E$ are either dominated by or equivalent to at least one $x \in X_E$. I.e., for each nondominated point $z \in Z_N$ there exists at least one $x \in X_E$ such that $z(x) = z$.*

   - *[13] A* minimal complete set *X_{E_m} is a complete set without equivalent solutions. Any complete set contains a minimal complete set.*
   - *The* maximal complete set *X_{E_M} is the complete set including all equivalent solutions, i.e all $x \in X \setminus X_{E_M}$ are dominated.*

As indicated by intractability of (BAP) the number of equivalent efficient solutions can be large. Indeed, it is known that for some instances of the (BAP) all feasible solutions are efficient. In that case, it is impossible to design an efficient algorithm for computing the maximal complete set X_{E_M}. This is the motivation for Hansen's introduction of the notion of minimal complete set X_{E_m}.

Supported and nonsupported efficient solutions can also be classified using the definitions of Hansen. Thus we can talk about

- Complete sets of supported and nonsupported efficient solutions.
- Minimal and maximal complete sets of supported and nonsupported efficient solutions X_{SE_m}, X_{NE_m}, X_{SE_M}, and X_{NE_M}.
- Extremal and non-extremal supported efficient solutions X_{SE1_m}, X_{SE2_m},X_{SE1_M} and X_{SE2_M}.

Published papers are sometimes vague about the ability of proposed algorithms. Some authors claim that their algorithm can enumerate "all" efficient solutions, but do not mention whether that includes equivalent solutions, or even solutions in X_{SE2}. To state clearly the characteristic of existing algorithms, it is important to give the class of efficient solutions that an algorithm computes.

1.3 Literature

According to [5], early papers on multiple objective assignment problems only deal with supported efficient solutions, using convex combinations of objective functions, or goal programming. Some algorithms to determine a complete set X_E have been proposed: [14] assume that the efficient solutions are connected by simplex pivots. [21] has shown this assumption is false. [22] proposed an operational procedure for the BAP derived from the two phase method. However, this procedure is very time consuming (see [20]). Consequently, approximation methods have been proposed for computing sub-optimal solutions with a reasonable computing time. Two methods have been proposed for the assignment problem: an extension of simulated annealing to deal with multiple objectives, called the MOSA method [21], and a population-based heuristic using path-relinking [9].

2 The Two Phase Method

The two phase method is a general framework for solving MOCO problems [22], although it has never been applied for problems with more than two objectives. The main idea is to use efficient algorithms for the single objective problem to compute efficient solutions. As efficient algorithms for single objective problems are problem specific it is necessary to preserve constraint structure of the problem throughout the solution procedure. It is therefore, for example, not possible to add constraints on objective function values (as is done in multiobjective integer programming methods, e.g., [15, 6]) to search for efficients solutions.

In Phase 1 supported efficient solutions are computed. It is based on Geoffrion's theorem [10] stating that an optimal solution of (BAP$_\lambda$) with positive weights is efficient.

In Phase 2, information from supported efficient solutions identified by Phase 1 is used to reduce the search space for computing the nonsupported efficient solutions. In implementations, Phase 2 is generally enumerative.

In the biobjective case, Phase 2 explores the triangles defined by two adjacent supported solutions (Figure 2) in the objective space. To search for nonsupported efficient solutions in an effective manner, lower bounds, upper bounds, reduced costs, etc. are usually employed.

The method has first been proposed for the biobjective assignment problem by [22]. Some experimental results have been published by [20]. But the results obtained by [3] using an MIP solver have shown more efficient solutions. This put the validity of the implementation in question. In fact, the method of [22] is not wrong but it does not consider all the possible cases as we report in this paper.

In this paper, we first provide a full description of the two phase method for the biobjective assignment problem. Second, we introduce improved bounds and new algorithms for Phase 2, with a discussion on the distribution of nondominated points which yields a justification of the efficiency of the algorithms. As the numerical experiments

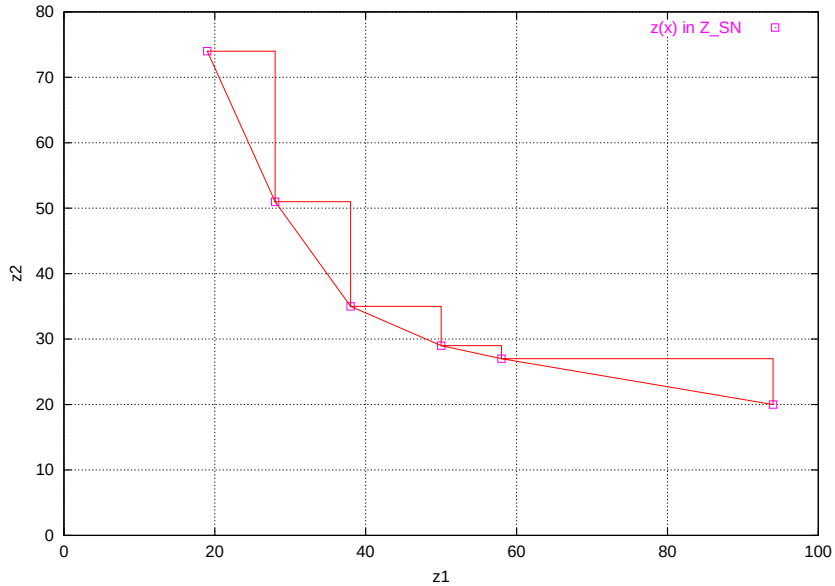

Figure 2: The exploration space is reduced to the interior of the triangles (instance `2AP10-1A20`).

show, the most efficient configuration of the procedure we propose for solving bi-objective assignment problems is based on a ranking algorithm, and outperforms CPLEX in terms of CPU time.

The remainder of the paper is structured as follows. Section 3 describes the Phase 1 algorithm. Section 4 develops some lower and upper bounds, and Section 5 presents several Phase 2 algorithms that allow finding minimal or maximal complete sets. All proposed algorithms are evaluated on a wide set of numerical instances. Nmerical results are reported and discussed in Section 6. The paper is concluded with a discussion on the distribution of solutions.

3 Phase 1: Determination of Supported Efficient Solutions

The Phase 1 algorithm described in this section is used unchanged for all algorithms described in the rest of the paper (see algorithms 1 and 2)

This phase determines the set X_{SE_M} by a dichotomic method. S denotes a set of efficient solutions already found. Initially, we set $S \leftarrow \{x^1, x^2\}$, the two lexicographic optimal solutions corresponding to $\text{lexmin}_{x\in X}(z^1(x), z^2(x))$ and $\text{lexmin}_{x\in X}(z^2(x), z^1(x))$, respectively.

3.1 Finding Lexicographic Extreme Efficient Solutions

To determine x^1 and x^2, we first compute one optimal solution (using procedure `solveAP` in algorithms 1 and 2) for each of the two single objective problems ($x^{1'}$ for $\min_{x\in X} z^1(x)$ and $x^{2'}$ for $\min_{x\in X} z^2(x)$) by solving (BAP$_{(1,0)}$) and (BAP$_{(0,1)}$). These solutions may be only weakly efficient.

While x^1 and x^2 can be found by solving $\min\{z^2(x) : x \in X,\ z^1(x) \leq z^1(x')\}$ and $\min\{z^1(x) : x \in X,\ z^2(x) \leq z^2(x')\}$, respectively, or by enumerating all optimal solutions of (BAP$_{(1,0)}$) and (BAP$_{(0,1)}$), these techniques might be time consuming in practice, because a single objective assignment problem may have an exponential number of optimal solutions and (BAP$_\lambda$) with a constraint derived from an objective function is $\mathcal{NP}$-hard [16].

We designed the following procedure, which is hereafter illustrated for the first objective. It is possible to find x^1 by solving an additional (BAP$_\lambda$) with the weight vector $(1, 0)$ slightly modified. Because $c^k_{ij} \in \mathbb{N}$ we know $Z \subset \mathbb{N}^2$. If we use weights defined by the normal to the line through $(z^1(x^1), z^2(x^1))$ and $(z^1(x^1) + 1, -1)$,

i.e., $\lambda^1 = z^2(x^1) + 1$ and $\lambda^2 = 1$, the optimal solutions of the corresponding (BAP$_\lambda$) are optimal solutions of lexmin$_{x \in X}(z^1(x), z^2(x))$. Note that we assume all c_{ij}^k to be integer.

x^2 is found the same way by using $x^{2'}$ and solving problem (BAP$_\lambda$) with weights $\lambda^1 = 1$ and $\lambda^2 = z^1(x^2)+1$.

3.2 Findind X_{SE1_M} and X_{SE2_M}

During the dichotomic search, the solutions of S are sorted by z^1 increasing. We will consider two consecutive solutions x^r and x^s according to this order, that are not equivalent, i.e., $z^1(x^r) < z^1(x^s)$ and $z^2(x^r) > z^2(x^s)$. A weighted sum problem (BAP)$_\lambda$ with $\lambda^1 = z^2(x^r) - z^2(x^s)$ and $\lambda^2 = z^1(x^s) - z^1(x^r)$ is solved, and all optimal solutions are enumerated using the algorithm of [7] (using procedure `enumerate` in algorithm 2). Initially, we will choose $x^r = x^1$ and $x^s = x^2$.

Let $\{x^t, t \in T\}$ be all optimal solutions of (BAP$_\lambda$), where T is an index set such that $|T|$ is the number of optimal solutions of (BAP$_\lambda$). Two cases can occur:

a) $\{z(x^t), t \in T\} \cap \overline{z(x^r)z(x^s)} = \emptyset$. All solutions x^t are new supported efficient solutions and we add them to S. Then we compute the minimum and maximum of $z^1(x)$ in $\{x^t : t \in T\}$. Let x^{t_1} and x^{t_2} be the solutions where the minimum and maximum are attained. Two new weighted sum problems are considered, one defined by x^r and x^{t_1} and one defined by x^{t_2} and x^s (using procedure `solveRecursion` in algorithms 1 and 2). It is not necessary to consider a scalarized problem defined by two solutions of $\{x^t, t \in T\}$ because the weight vector λ will be the same as that defined by x^r and x^s and therefore no new solutions will be found.

b) $\{z(x^t), t \in T\} \subset \overline{z(x^r)z(x^s)}$. All solutions x^t are supported efficient solutions and we add the new solutions to S, but no new scalarized problem is generated.

Phase 1 stops if no new weighted sum problems (BAP$_\lambda$) have to be solved.

At the end of Phase 1, $S = X_{SE_M}$. Note that without using an enumerative algorithm, we will in general only find a minimal complete set X_{SE1_m} and possibly some other supported efficient solutions. The numerative algorithm is necessary to be sure to find a set X_{SE2_m}.

4 Bounds

In Phase 2 we will try to find $x \in X$ such that $z(x)$ is in the triangle defined by two consecutive nondominated supported points $z(x^r)$ and $z(x^s)$ in the objective space. We will use lower and upper bounds on objective values of single objective assignment problems as stopping conditions of our algorithms.

4.1 Lower Bounds

Lower bounds avoid the exploration of solutions that cannot be efficient. In all Phase 2 algorithms presented until now the lower bound of [22] is used. Let $x \in X$ and let C be the matrix of objective function coefficients of a single objective AP. We are interested in the objective value that results from fixing a variable $x_{i^*j^*} = 1$. In the algorithm C will be either C^1, C^2 or $\lambda^1 C^1 + \lambda^2 C^2$ with $\lambda^1 > 0$ and $\lambda^2 > 0$, or a square submatrix of any of these.

The bounds differ depending on whether x is optimal for the single objective problem or not.

4.1.1 Lower Bound from an Optimal Solution

Suppose that x is optimal for the single objective assignment problem $\min_{x \in X} Cx$. Thus we can find a reduced cost matrix $\bar{C}$ for this problem with only nonnegative entries. This can be, e.g., the reduced cost matrix given by the Hungarian method. We suppose that we will impose $x_{i^*j^*} = 1$, but in x there are i_t, j_t such that $x_{i_t j^*} = x_{i^* j_t} = 1$. Consequently, in the optimal solution of the problem with the fixed variable the value of at least one variable in row i_t and one variable in column j_t, other than $x_{i_t j^*}$ and $x_{i^* j_t}$, will be 1.

- If these two variables coincide, we will have $x_{i_t j_t} = 1$ and the change (increase) of the objective value will be at least $\bar{c}_{i_t j_t}$.

- Otherwise the increase will be at least

$$\gamma = \min_{j \neq j^*} \bar{c}_{i_t j} + \min_{i \neq i^*} \bar{c}_{i j_t}.$$

Therefore, a lower bound on the objective value after fixing the variable $x_{i^* j^*} = 1$ is

$$\alpha_1 = Cx + \bar{c}_{i^* j^*} + \min \{\bar{c}_{i_t j_t}, \gamma\}. \tag{1}$$

If there is more than one optimal solution, we can compute a lower bound from each of these solutions and keep the biggest value of α_1.

4.1.2 Lower Bound from a Non-optimal Solution

Suppose that x is not an optimal solution of $\min_{x \in X} Cx$. Let $\bar{C}$ be any reduced cost matrix with respect to some basis corresponding to x or a dual solution, which verifies $\bar{c}_{ij} = c_{ij} - u_i - v_j = 0$ for all (i, j) with $x_{ij} = 1$. Here, u_i and v_j are the usual dual variables corresponding to the constraints. Any such reduced cost matrix $\bar{C}$ contains some negative entries.

Since the objective function change resulting from fixing a variable to 1 may be negative, the lower bound on the change is computed using the minimal elements of all rows and columns of $\bar{C}$. Consequently, a lower bound on the objective value after fixing the variable is

$$\alpha_2 = Cx + \bar{c}_{i^* j^*} + \max \left\{ \sum_{i \neq i^*} \min_{j \neq j^*} \bar{c}_{ij}, \sum_{j \neq j^*} \min_{i \neq i^*} \bar{c}_{ij} \right\}. \tag{2}$$

Because of the negative coefficients, this lower bound can be less efficient than (1) in practice.

4.2 Upper Bounds

Let x^r and x^s be two consecutive supported efficient solutions and let λ be the weight vector for which both x^r and x^s are optimal solutions of (BAP$_\lambda$). We derive upper bounds on the objective value $\lambda^1 z^1(x) + \lambda^2 z^2(x)$ for efficient solutions x with $z(x)$ in the triangle defined by $z(x^r)$ and $z(x^s)$. Let $\Delta(x^r, x^s)$ denote (the interior of) that triangle.

In the Phase 2 algorithms we will explore each triangle defined by two adjacent supported efficient solution separately. While the algorithms differ in their exploration strategy, all have in common a complete enumeration of bands in the triangle, i.e., areas in the triangle between the line segment $\overline{z(x^r)z(x^s)}$ and a line parallel to that. This parallel line will be defined by the upper bounds β_i as $\{z \in \mathbb{R}^2 : \lambda^T z = \beta_i\}$.

We will use information from points already explored to reduce the bands by shifting the parallel line towards $\overline{z(x^r)z(x^s)}$. We have used different bounds in order to obtain different complete sets. To illustrate the bounds, we will use as an example a triangle of the instance 2AP30-1A20 (see Section 6) defined by the supported efficient solutions x^r, x^s with $z(x^r) = (82, 63)$ and $z(x^s) = (99, 51)$. The nondominated points in this triangle are (86,61), (88,60), (91,59), (92,58), (93,56), (95,55), (97,54), (98,52). These points are shown, e.g., in Figure 3.

4.2.1 Upper Bound of Tuyttens et al. [20]

In Figure 3, the eight nondominated points and the area they dominate are shown. We can see that parallel to the line segment $\overline{z(x^r)z(x^s)}$ there is a line, so that all points of the triangle dominated by a point on that line are also dominated by one of the nondominated points. Because this line is parallel to $\overline{z(x^r)z(x^s)}$ only one point is necessary to define it. If we sort the nondominated points (including $z(x^r)$ and $z(x^s)$) by z^1 increasing, this line passes through one of the local nadir points defined by two adjacent solutions.

In the algorithms we work with a set of potentially efficient solutions X_{PE} identified so far. We denote Z_{PN} the corresponding set of potentially nondominated points. Let $\left\{x^i : 0 \leq i \leq q\right\}\}$ be X_{PE} sorted by z^1 increasing. Let $\gamma = \max_{i=1}^{q} \{\lambda^1 z^1(x^i) + \lambda^2 z^2(x^{i-1})\}$. Thus an upper bound on any nondominated point in the triangle is

$$\beta_0 := \max \left\{\gamma, \lambda^1 z^1(x^0) + \lambda^2 z^2(x^r), \lambda^1 z^1(x^s) + \lambda^2 z^2(x^q)\right\}.$$

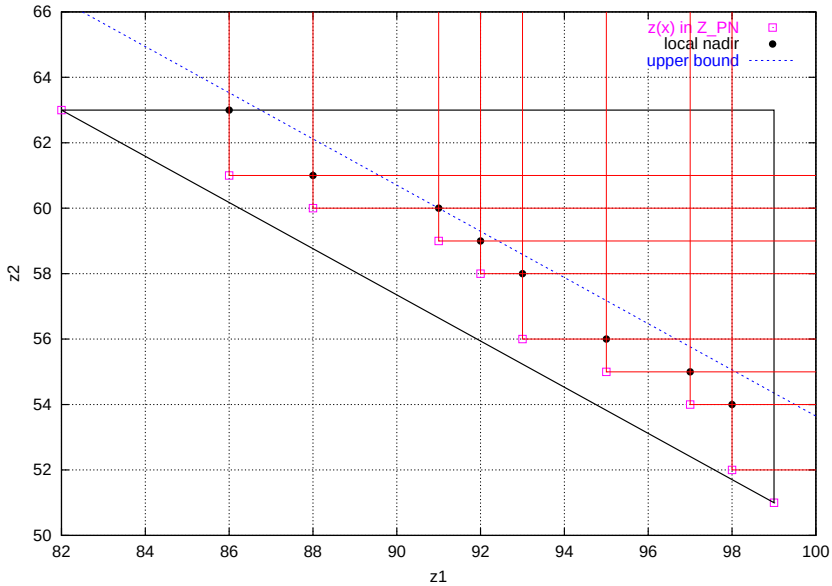

Figure 3: The bound of [20] defines this dotted line which is located in the dominated area.

In Figure 3 the maximum is found for the local nadir point (91,60) with value 2112 (an increase of 57 from x^r and x^s).

As all the solutions $x \in X, z(x) \in \Delta(x^r, x^s)$ with $\lambda^1 z^1(x) + \lambda^2 z^2(x) \geq \beta_0$ are dominated, enumeration of all $x \in X$ with $z(x)$ between the two lines yields all nonsupported efficient solutions with $z(x)$ in the triangle, including the equivalent ones. After exploring all triangles, we will find the maximal complete set X_{E_M}.

4.2.2 Improved Upper Bound

With the bound of [20] we neglect the fact that $z(x)$ has integer coordinates for all $x \in X$. We can use this fact to improve the upper bound. If we translate the line defined by the bound β_0 towards the supported nondominated points the first point of integer coordinates we can find is either a point which is one unit down and left of a local nadir point, or one of the potentially nondominated points (Figure 4).

Let $\delta_1 = \max_{i=0}^{q}\{\lambda^1 z^1(x^i) + \lambda^2 z^2(x^i)\}$ be the maximum weighted sum objective for the potential efficient solutions and $\delta_2 = \max_{i=1}^{q}\{\lambda^1(z^1(x^i) - 1) + \lambda^2(z^2(x^{i-1}) - 1)\}$ the maximum for the points one unit down and left of the local nadir points. Then the improved bound is defined by

$$\beta_1 := \max\left\{\delta_1, \delta_2, \lambda^1(z^1(x^0) - 1) + \lambda^2(z^2(x^r) - 1), \lambda^1(z^1(x^s) - 1) + \lambda^2(z^2(x^q) - 1)\right\}.$$

In the following, this bound will be called *Bound 1*.

In Figure 4 the maximum is found for the nondominated point (91,59) and has value 2095 (an increase of 40 from x^r and x^s).

All solutions $x \in X, z(x) \in \Delta(x^r, x^s)$ with $\lambda^1 z^1 + \lambda^2 z^2 > \beta_1$ are dominated. Therefore, enumeration of all feasible solutions with $(z^1(x), z^2(x))$ between the lines $\overline{z(x^r)z(x^s)}$ and $\{z : \lambda^1 z^1 + \lambda^2 z^2 = \beta_1\}$ (inclusive) yields all nonsupported efficient solutions of the triangle including equivalent ones. Thus, after exploring all triangles, we will find the maximal complete set X_{E_M}.

4.2.3 An Improved Bound to Find a Complete Set

If only a complete set is aimed at as output of the algorithm, Bound 1 can be revised to yield an improved algorithm. In this case, it is not necessary to consider the potentially efficient solutions when computing β_1, see Figure 5.

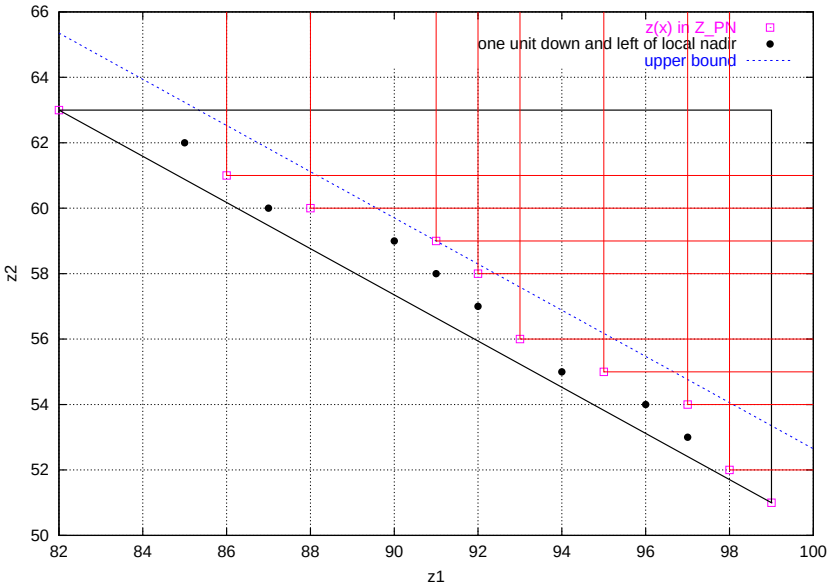

Figure 4: Bound 1: A first improvement of the upper bound of [20]

The bound is defined by

$$\beta_2 := \max\left\{\delta_2, \lambda^1(z^1(x^0)-1)+\lambda^2(z^2(x^r)-1), \lambda^1(z^1(x^s)-1)+\lambda^2(z^2(x^n)-1)\right\}.$$

In the following, this bound will be called *Bound 2*.

Each feasible solution $x \in X, z(x) \in \Delta(x^r, x^s)$ with $\lambda^1 z^1(x) + \lambda^2 z^(x) > \beta_2$ is dominated or equivalent to one of the potentially efficient solutions. Therefore, enumeration of all feasible solutions with $(z^1(x), z^2(x))$ between the lines $\overline{z(x^r)z(x^s)}$ and $\{z : \lambda^1 z^1 + \lambda^2 z^2 = \beta_1\}$ (inclusive) yields all nonsupported efficient solutions in that band. In addition we already have some potentially efficient solutions outside the band, which may be proven efficient at the end of the exploration. Consequently, after exploring all triangles this way, we will find a complete set X_E.

Further reduction of the bound is not possible, as we could miss some nondominated points, and consequently keep some dominated ones as nondominated: β_2 is a tight bound for an exact method with an enumerative exploration strategy.

In Figure 5 the maximum is found for the point (90,59) with value 2083 (an increase of 28 from x^r and x^s). That means a complete set of efficient solutions in this triangle is obtained with at most 1.36% increase in the value of z^λ.

5 Phase 2: Finding Nonsupported Efficient Solutions

5.1 General Principle of the Original Method

Let x^r and x^s be two adjacent supported efficient solutions that are not equivalent. And let λ be such that both x^r and x^s are optimal solutions of (BAP$_\lambda$). In Phase 2 we search for feasible solutions x with $z^1(x^r) < z^1(x) < z^1(x^s)$ and $z^2(x^r) > z^2(x) > z^2(x^s)$.

Obviously, all such solutions are in the triangle $\Delta(x^r, x^s)$. [22] propose an enumerative exploration method to search for these solutions. The idea is to find nonsupported efficient solutions by imposing assignments, i.e., setting $x_{ij} = 1$ in (BAP$_\lambda$), and solving (reoptimizing) the reduced assignment problem that results. In other words, we search for nonoptimal feasible solutions x of (BAP$_\lambda$) starting from x^r and x^s by modifying these solutions.

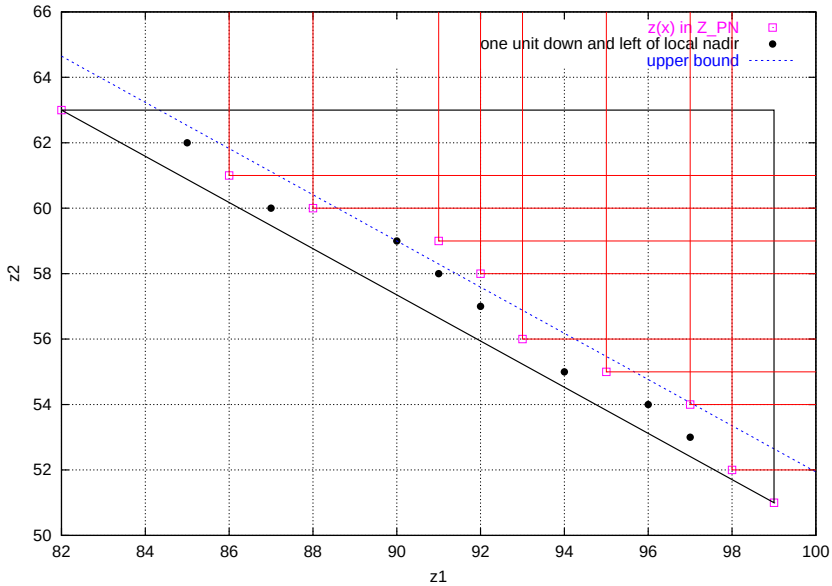

Figure 5: Bound 2.

The lower bounds α_1 and α_2 of Section 4.1 are used to check if imposing an assignment can lead to a solution in the triangle $\Delta(x^r, x^s)$. Otherwise imposing $x_{ij} = 1$ is not necessary. Let L denote the list of pairs (i, j) that are candidates for variable fixing. By individually imposing each of the assignments of L, we partially explore the triangle. It is well-known ([22] and [4]) that it is necessary to simultaneously impose more than one assignment of L. But no complete description of the procedure for this simultaneous variable fixing has been published.

During this search, the feasible solutions generated are stored in a list of potentially efficient solutions X_{PE} . Each new solution x is compared to the solutions of X_{PE}, and X_{PE} is updated if necessary. At termination of the search, i.e., if all solutions $x \in \Delta(x^r, x^s)$ not yet enumerated are guaranteed to satisfy $\lambda^1 z^1(x) + \lambda^2 z^2(x) > \beta_i$ for one of the upper bounds of Section 4.2, X_{PE} is a complete set of nonsupported efficient solutions in the triangle.

In the following, we present a complete description of a Phase 2 procedure which follows the original principle. We search in a depth-first manner, fixing variables according to list L one by one until we are sure that no further efficient solutions can be reached.

5.2 Phase 2: A Complete Description

5.2.1 Imposing and Forbidding Assignments of L

The main goal of this subsection is to show how finding the same solutions repeatedly by imposing combinations of assignments can be avoided.

We will call a set of variables $\{(i_k, j_k) | k \in K\}$ *compatible assignments* if imposing $x_{i_k j_k} = 1$ for all $k \in K$ does not violate the constraints of (BAP). To avoid double indexing we will write x_k below instead of $x_{i_k j_k}$. Before imposing an additional assignment, we will verify that it is compatible with the ones that have already been imposed.

Suppose $L = \{x_1, x_2, x_3\}$ and that all assignments are compatible. We propose a depth-first search. Because imposing $\{x_1, x_2\}$ and $\{x_2, x_1\}$ will yield the same solution, we can suppose that L is sorted and that the indices are in the increasing order. With $L = \{x_1, x_2, x_3\}$ we have the following possible sets of variables that can be fixed to 1: $\{x_1\}, \{x_1, x_2\}, \{x_1, x_2, x_3\}, \{x_1, x_3\}, \{x_2\}, \{x_2, x_3\}, \{x_3\}$.

We will find all solutions containing x_1 in the four first sets. By imposing $\{x_2\}$, we may find a solution containing x_1. However, we have already visited it. We can avoid that by fixing $x_1 = 0$ in the fifth set of variables, i.e., forbidding x_1. This can be done by replacing the cost of x_1 by a very high number. In the same way, we know

that after the three first sets, we have all solutions containing x_1 and x_2, so we can forbid x_2 in the fourth set of assignments $\{x_1, x_3\}$.

In general, suppose we have a set of variables $S = \{x_{i_1}, x_{i_2}, \ldots, x_{i_r}\}$ to be fixed to 1. Then we can forbid assignments $x_i,\ i < i_r$, i.e., fix $x_i = 0$ for $i < i_r$ if $x_i \notin S$.

5.2.2 The List of Assignments L

Initially, we use

$$L = \left\{(i,j) | \bar{c}^{\lambda}_{ij} > 0\right\},$$

where $\bar{C}^{\lambda}$ denotes the reduced cost matrix obtained by the Hungarian method for the problem (BAP_{λ}) for which x^r and x^s are optimal, i.e., the same initial list of assignments as [22].

We can delete assignments from L by testing if imposing them yields a point outside $\Delta(x^r, x^s)$, as proposed by [22]. We start from the two supported solutions x^r and x^s. To do this, we use the lower bound described in Section 4.1. There will be three tests:

Test 1: Lower bound l_{λ} on z^{λ} compared with $z^{\lambda}(x^r) = z^{\lambda}(x^s)$ (with the reduced costs obtained by the Hungarian method for z^{λ}).

l_{λ} is the highest of the values of α_1 obtained from x^r and x^s in (1). The assignment can be deleted from L, if l_{λ} shows that a solution containing this assignment will be beyond the line parallel to $\overline{z(x^r)z(x^s)}$ containing the point $z = (z^1(x^s) - 1, z^2(x^r) - 1)$. The increase of z^{λ} from the supported efficient solutions to z is

$$\lambda^1(z^1(x^s) - 1) + \lambda^2(z^2(x^r) - 1) - \lambda^1 z^1(x^r) - \lambda^2 z^2(x^r) = \lambda^1\lambda^2 - \lambda^1 - \lambda^2.$$

Consequently, the assignment can be deleted if

$$l_{\lambda} > z^{\lambda}(x^r) + \lambda^1\lambda^2 - \lambda^1 - \lambda^2.$$

Test 2: Lower bound l_1 on z^1 compared with $z^1(x^r)$ (with the reduced costs obtained by the Hungarian method for z^1 if $x^r = x^1$).

l_1 is computed either as α_1 in (1) or α_2 in (2), depending on whether x^r is an optimal solution of (BAP$_{(1,0)}$) or not. The assignment can be deleted from L, if l_1 shows that the solution will be to the right of a veryical line through $z(x^s)$, i.e., if

$$l_1 \geq z^1(x^r) + \lambda^2.$$

Test 3: Lower bound on z^2 compared with $z^2(x^r)$ (with the reduced costs obtained by the Hungarian method for z^1 if $x^s = x^2$).

l_2 is computed either as α_1 in (1) or α_2 in (2), depending on whether x^s is an optimal solution of (BAP$_{(0,1)}$) or not. The assignment can be deleted if l_2 shows that the solution will be above a horizontal line through $z(x^r)$, i.e., if

$$l_2 \geq z^2(x^s) + \lambda^1.$$

Once the tests are completed and L has been updated accordingly, we can begin the depth-first search. Recall that to be sure to find all efficient solutions it is necessary to do an enumeration of all optimal solutions of every reduced problem. However, for each of the problems with fixed variables we may find many solutions which may even be repeated several times.

Alternatively, we can to add to L the assignments such that $\bar{c}^{\lambda}_{ij} = 0$, because this way assignments with reduced costs zero will be imposed (and forbidden) in the combinations of assignments, and no enumeration will be necessary. Although the number of combinations will be larger, numerical evidence suggests that avoidance of redundancy is worth that price, and this alternative is more efficient computationally. Moreover, we will consider the obtained solutions one by one, and it will be easy to reuse the lower bounds of Section 4.1 during the depth-first search as it wil be shown in the next section.

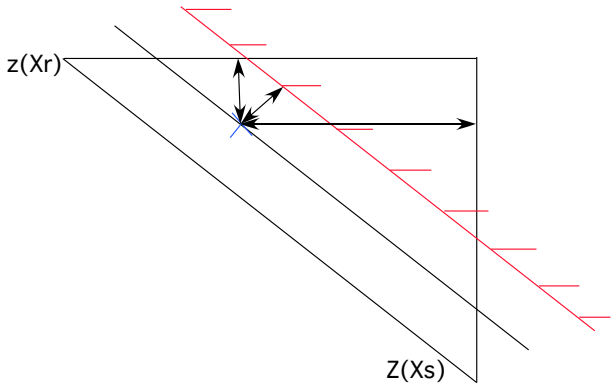

Figure 6: Test before adding an assignment.

5.2.3 Stopping Conditions

To solve a reduced assignment problem after fixing a variable, we can either use the cost matrix or the reduced cost matrix (given by the preceeding optimization). However, with the reduced cost matrix we can use a lot of zeros from the preceeding solution, and solve the problem faster. Note that each time we add an assignment (say x'' is the new optimal solution of the reduced problem resulting from imposing an assignment in solution x'), $z^\lambda(x'') \geq z^\lambda(x')$. Consequently, if $z^\lambda > \beta_i$ we cannot find new efficient solutions by imposing more assignments. This is the first stopping condition.

Note that finding a solution outside of $\Delta(x^r, x^s)$ with a reoptimisation does not mean that a subsequent solution with more imposed assignments will not be in $\Delta(x^r, x^s)$ located inside the triangle. To find a solution outside of the triangle is not a stopping condition.

However, even if $z^\lambda(x'') \leq \beta_i$ not all possible additional variable fixings and reoptimizations are useful. We can use the lower bounds to test if adding further assignments will give only solutions outside of the triangle or above the upper bound β_i.

Note that x' is an optimal solution of a reduced problem (BAP$_\lambda$), but not necessarily of the reduced problem with objectives C^1 or C^2. Let l'_λ, l'_1, l'_2 denote the lower bounds on the objective function values z^λ, z^1 and z^2, respectively, of a solution resulting from fixing an additional variable. Reoptimisation can be avoided if l'_λ shows that we are sure to find a solution in the hashed area in Figure 6. Let β_λ be the current upper bound on z^λ obtained during the process. Thus, if

$$l'_\lambda > \beta_\lambda \tag{3}$$

there is no need to fix the variable.

In the same way, we can use lower bounds on z^1 and z^2 to see if a reoptimisation after variable fixing yields a solution outside of $\Delta(x^r, x^s)$. If one of these two lower bounds shows that this will be the case, all solutions obtained by fixing more variables to 1 will also be outside $\Delta(x^r, x^s)$.

Thus if

$$l'_1 \geq z^1(x^r) + \lambda^2 \quad \text{or} \tag{4}$$
$$l'_2 \geq z^2(x^s) + \lambda^1 \tag{5}$$

there is no need to fix the variable. So, before fixing a variable, we will check if one of the inequalities (3), (4), or (5) is verified. This is the second stopping condition.

5.2.4 Description of the Algorithm

We consider one triangle given by two adjacent supported efficient solutions at a time. If $\lambda^1(z^1(x^s) - 1) + \lambda^2(z^2(x^r) - 1) \leq \lambda^1 z^1(x^r) - \lambda^2 z^2(x^r)$ it is not necessary to continue.

Otherwise we compute the list of assignments L which will be used for the depth-first search: $L := \{(i,j) : \bar{c}^{\lambda}_{ij} \geq 0\}$. Tests 1, 2, and 3 are performed for all of these variables (using `isTest_123` in algorithm 3). If all pairs with strictly positive reduced cost are deleted by the tests, it is not necessary to continue.

Otherwise the depth-first search is carried out. L is sorted by increasing value of α_1 obtained with Test 1, see (1). The depth-first search is a recursive function (see procedure `impose` in algorithms 3 and 4) and fixing variables to 0 is naturally integrated as shown in Section 5.2.1. Before adding an assignment, compatibility with the other fixed variables is checked (using procedure `compatible` in algorithm 4).

After each reoptimization, giving a solution x', we verify that its value does not violate the upper bound (first stopping condition). In that case, we verify if it is necessary to update X_{PE} (using procedure `isInTriangle`, `isDominated` and `removeDominatedSolution` in algorithm 4) and consequently the upper bound (using procedure `computeUpperbound` in algorithm 4). Finally, we check if it is necessary to add the next compatible assignment by checking the lower bounds (second stopping condition using procedure `NeedtoFixVariable`). If it is, we reiterate the process.

5.3 Seek & Cut Algorithm: An Improvement of Phase 2

5.3.1 Idea of the Algorithm

In the original Phase 2 algorithm we start with $X_{PE} = \emptyset$. Therefore the initial upper bound may be bad, and its decrease during the algorithm slow. The idea of the *Seek and Cut algorithm* is to search for X_{PE} solutions using a fast heuristic to obtain a good approximation of the nondominated frontier. These solutions are sorted by triangle, for which the exact Phase 2 algorithm therefore starts with a nonempty X_{PE} and an improved upper bound.

5.3.2 The Heuristic by Gandibleux et al. [9]

The heuristic proposed by [9] is a population-based heuristic using a path-relinking operator. The heuristic uses three operators – local search, crossover, and path relinking – performed on a population composed only of potentially efficient solutions. The initial population is a subset of X_{SE}, e.g., X_{SE1_m}. The heuristic can be used with different parameters to find approximations of X_E of varying quality with respect to CPU time. The approximation quality is very sensitive to the path relinking operator. We have used three different versions of path relinking.

Path-relinking generates new solutions that connect potentially efficient solutions starting from one solution (the initiating solution), generating a path through the solution space using a neighborhood structure that leads toward the other (the guiding) solution [11].

A path-relinking operator starts by randomly selecting $I1$ and $I2$, two individuals in the current potentially efficient population of solutions. Because both are potentially efficient, both can be the guiding solution. Let $I1$ be the initiating solution and $I2$ the guiding solution. The path-relinking operator generates a path $I1 = I_0, I_1, \ldots, I2$, such that the distance between I_i and $I2$ decreases along the path, where distance is defined as the number of different positions for tasks assigned in I_i and $I2$. At each iteration, a list LDA (list of different assignments) reports the tasks assigned to different positions in solutions I_i and $I2$, and the neighbourhood is defined by a simple swap: A new solution is built by selecting one task in LDA, and assigning this task to its final position j in the guiding solution $I2$. To maintain the solution feasible, the task currently in position j in I_i is moved to j', the current position of the selected task. The three variations we consider are:

- Pure random path relinking: A new solution is built by randomly selecting one task from LDA. If the solution is a promising solution (i.e. a solution which is not dominated by one of the local nadir points defined by adjacent supported points), a local search is performed. We will call this procedure $(PR1)$.

- Selected random path relinking. We check each solution which reduces the distance between the initial (or current) and the guiding solution, to check if some are (possibly new) potentially efficient solutions. The set of potentially efficient solutions is updated if necessary. Then we choose randomly one of the potentially efficient solution of the neighborhood of the current solution to continue the path relinking. If there is no potentially efficient solution in the neighborhood the next solution is selected purely randomly. If the new solution is a promising solution, we perform a local search. This procedure is called $(PR2)$.

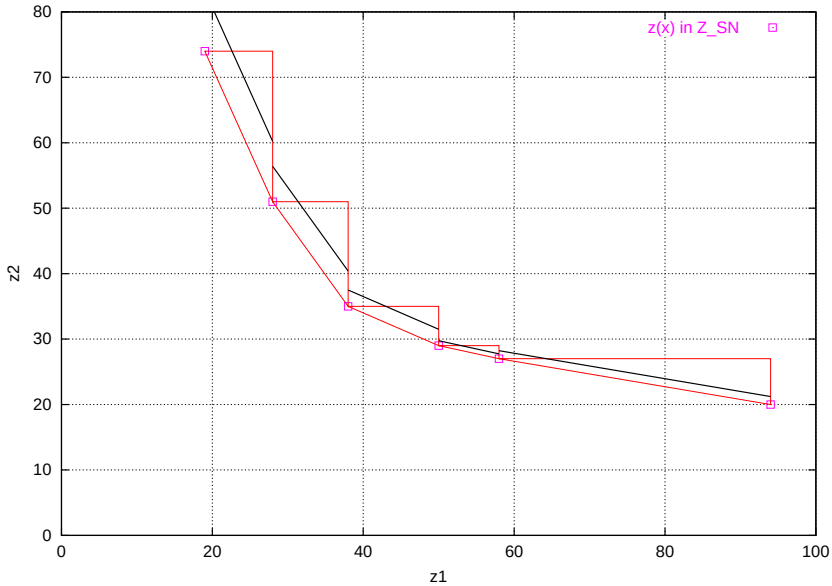

Figure 7: The exploration space is reduced by the new initial bounds (instance 2AP10-1A20).

- Selected random path relinking with local search. This method is the same as $(PR2)$, except that for all the solutions in the neighborhood of the current solution before the choice, if some are promising solutions, we perform a local search for the promising ones. We will call this $(PR3)$.

The heuristic proposed by [9] can be used with several stopping conditions. We have tried the following: 250.000 generations for the heuristic (stopping condition $(N1)$), and stop after 100.000 generations without improvement of X_{PE} (stopping condition $(N2)$).

Therefore, we have six parameter settings $(PR*)(N*)$, some of which are computationally expensive (in particular $(PR3)$ and $(N2)$). However, the time needed by the heuristic is negligible compared to the time needed by the phase 2 with variable fixing strategy.

5.3.3 Description of the Algorithm

First, we compute X_{SE_M} using Phase 1. The parameters for the heuristic are chosen and the heuristic is run. The set X_{PE} found by the heuristic is sorted by triangle and kept as initial set X_{PE} in Phase 2. We can now consider each triangle separately and start Phase 2. Using the initial set X_{PE} we can compute a good initial upper bound (Figure 7). From here the Phase 2 proceeds as before.

5.4 Phase 2 Using a Ranking Approach

In this subection, we propose a different way of exploration of each triangle defined by adjacent supported efficient solutions in Phase 2. Essentially, we search for solutions with increasing value of z^λ until one of the upper bounds β_i is reached. This can be done using a ranking method, i.e., an algorithm that finds solutions in increasing order of their z^λ value.

This procedure is a natural choice in Phase 2: As it does not induce a modification of the problem structure it does not betray the original principle of the two-phase method. The advantages of a ranking algorithm rather than a variable fixing approach are that redundancy is avoided and that the exploration is ordered, because of the monotonicity with respect to z^λ values.

The strategy requires an efficient ranking algorithm. For the assignment problem, such an algorithm has been proposed by [2]. Essentially, this algorithm is an application of the binary search tree algorithm published by [12].

In the algorithm it is necessary to compute the best and second best solution of a single objective problem. [2] have shown that the second best solution of a single objective AP is obtained by application of a shortest cycle to the permutation given the best solution. To obtain this cycle [2] generate a bi-partite graph on which shortest path algorithms are performed.

We consider the problem (BAP$_\lambda$) for which adjacent solutions x^r and x^s are optimal and we apply an algorithm to find the k-best solution iteratively. As in the other Phase 2 methods, for each new solution (obtained using procedure `ComputeNextK_bestSolution` in algorithm 5), we verify if it is neccessary to update X_{PE} and the upper bound. The procedure stops as soon as we first exceed an upper bound β_i. If we also apply the lower bounds α_i we avoid enumerating points outside the triangle $\Delta(x^r, x^s)$. Due to the ranking approach there is no need to fix variables to one, however, we will forbid variables whose inclusion in a solution leads to points outside the triangle by replacing their cost with a very high number.

While Tests 2 and 3 allow to avoid exploration outside the triangle, Test 1 does not reduce the number of solutions explored. However, the exclusion of assignments obtained with Test 1 implies that the graph generated to compute the second best solution is sparser, and then the ranking algorithm performs slightly faster.

The only stopping condition for this Phase 2 algorithm is given by the upper bound. Note that this way, as the exploration is naturally ordered, there is no need to remove solutions from X_{PE}. Each new solution given by the ranking, which is not dominated at this stage of the process is definitively efficient.

6 Numerical Experiments

6.1 Experimental Environment

A library of numerical instances for multiobjective combinatorial optimization problems is available on the internet at `www.terry.uga.edu/mcdm/`. The name of an instance provides the following characteristics: the number of objectives, the problem, the size n, the series, the objective type, and the range of objective function coefficients in $[0, value]$. For example, 2AP05-1A20 is a biobjective (2) assignment problem (AP) with 5×5 variables (05) from series 1 (1); the coefficients of the objective function are generated randomly (A) in the range $[0, 20]$ (20). The coefficients of both objective functions are all generated independently of one another, following a uniform distribution.

The Instances 2AP05-1A20 through 2AP50-1A20 have been used in [20]. We have completed these instances to obtain a series with size from $n = 5$ to $n = 100$ with one instance for each size. In addition, for sizes $n = 10$ to $n = 100$ with an increment of 10, we have generated ten instances each with objective function coefficients in the range $[0, 20]$ and one instance each with ranges $[0, 40]$, $[0, 60]$, $[0, 80]$, $[0, 100]$.

The computer used for the experiments is equiped with a PowerPC G4 1Ghz processor with 512 MB of RAM, and runs under the OS X operating system. The algorithms have been implemented in C. The binary has been obtained using the compiler gcc with the optimizer option -O3. The exceptions are for a comparison of the Two Phase method with ranking, using bound β_2 with CPLEX (on a Pentium 4 3.4 GHz processor with 4GB RAM under Red Hat Linux) and the results from [20] which we report for comparison (on a DEC 3000 Alpha).

We have used the Hungarian method to solve the single-objective assignment problems (an implementation with the successive shortest path algorithm has given worse results) and the algorithm of [7] to find all optimal solutions.

6.2 Analysis

We have analysed the performance of the original Two Phase method (with our depth-first search implementation of variable fixing), the Two Phase Method with all six variants of the Seek & Cut Algorithm, and the Two Phase method with ranking. In addition, we have tested the time needed to confirm optimality by providing a minimal complete set X_{NE_m} as input to Phase 2. Two variants of each method are obtained by using upper bound β_1 or β_2. Finally, we have used CPLEX as an MIP solver to generate a set X_{SE_m}. A dichotomic scheme has been designed ([8]): the solver is invoked for solving a (BAP_λ) with two additional constraints $z^1(x) < z^1(x^s)$ and $z^2(x) < z^2(x^r)$, where x^r and x^s are two optimal solutions of (BAP_λ), for reducing the feasible domain in objective space. The results on the first test series are given in Tables 2 and 3. The first six rows give the size of the

various efficient sets, the following the results of all 17 methods (indeed, the tuning $(PR3)(N2)$ has not been used with Seek & Cut with upper bound β_2 because the tuning $(PR3)(N1)$ is already too expensive). Finally, a row comparing the Two Phase method with ranking and bound β_2 with CPLEX, and the results from [20] are given.

6.2.1 Results of the Two Phase Method and Seek & Cut Algorithm

Whatever the used bound, CPU time used by the Two Phase method increases exponentially with the size of the problem. In addition the increase in CPU time with upper bound β_1 is larger than with β_2 when the size increases. In fact, at the end of the exploration of a triangle the difference between the objective value of the supported solution and the upper bound is often near zero (for $n > 20$). This implies that in most triangles we need only explore a slim band. Therefore, the improved bound obtained by Seek & Cut before enumeration is effective. This explains the difference in CPU time between the Two Phase method and the Seek & Cut algorithm. Note that the computational effort of using the heuristic only shows advantages over the original Two Phase method for size $n \geq 50$.

Recall that with upper bound β_1, we obtain the maximum complete set X_{E_M} and with upper bound β_2, we obtain a complete set. With both bounds, considering a minimal complete set contained in these sets we have obtained the same number of supported efficient solutions, with the same points in objective space Z_N as [3] with an MIP solver. We can see that the number of equivalent solutions can be large for the largest instances, and that the equivalent solutions are often nonsupported.

We also observe that the Seek & Cut method is sensitive to the quality of the nondominated frontier obtained by the heuristic. For example, in the instance 2AP100-1A20, with bound β_2, the improvement to the the two-phases method is a factor between 4 or 7. However, the CPU time for bound β_1 is less sensitive to the results of the heuristic and the gap between the Two Phase method and Seek & Cut is smaller (a factor 2 for the instance 2AP100-1A20). There is a limit to this, namely, if instead of the heuristic we provide a minimal complete set X_{NE_m} as input. We then test the time used by the Two-Phase method to confirm the results (with bound β_2) or to confirm and find the equivalent solutions (with bound β_1).

The results with bigger range of objective coefficients have given worse results, because the elimination of solutions from enumeration is more effective with a small range of values (resulting in small triangles) than with a big range, when the triangles are larger. The difference between the bounds relative to the objective values is smaller in the case of a bigger range.

6.2.2 Results of the Method Using a Ranking Algorithm

We have done some more tests on the method using ranking on the series of 10 instances of size $10, \ldots, 100$ with range $[0, 20]$. These results are reported in Table 5. Here we report the minimum, maximum and average of time needed to solve the 10 instances of each size for Seek & Cut with parameters (PR2)(N1) and bound β_2 and ranking with both β_2 and β_1.

The CPU times obtained with the ranking method are better by a factor 10 to 100. Moreover, this method is less sensitive to the bound used and the range. This can be explained by the fact that no solution is repeated and we never consider solutions which exceed the final bound of the exploration. The distribution of the solutions in the objective space also provides justification that the ranking method is a better approach for solving the (BAP). The two phase method with ranking has been compared with CPLEX and it seems that one important advantage of the method is that it is robust with respect to the range of objective function coefficients.

Note that we have not tried to improve the results of the ranking method by using a heuristic to improve the initial bound because the time needed by the heuristic and by the ranking method is not very different.

6.2.3 Distribution of Objective Values and its Consequences

The huge improvement in CPU time obtained by the ranking methods can be explained by the huge difference in the number of enumerated solutions as compared to the original Two Phase Method (with or without heuristic). We have observed that near the boundary of the convex hull of Y there are relatively few feasible points. The number of feasible points increases exponentially with increasing objective values. Indeed, we can conjecture that the distribution of objective function values for a large size follows a normal distribution.

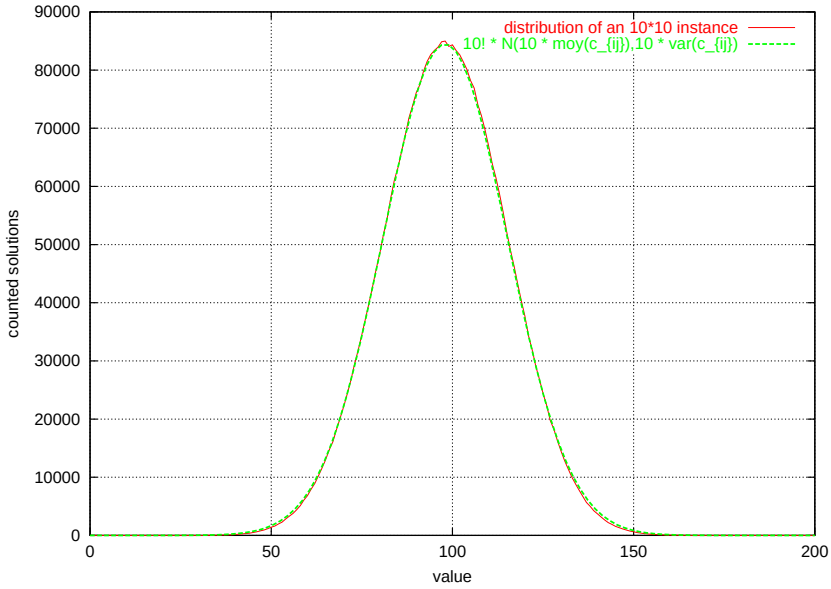

Figure 8: Counting solutions of an assignment problem obtained by enumeration.

Experimentally, we have enumerated the objective function values of all feasible solutions of an AP with $n = 10$. The distribution matches a normal distribution closely. In fact, the mean and variance of a normal distribution fitted to the experimental distribution matches those predicted by the central limit theorem exactly, see Figure 8.

Consequently, for large n, the distribution of a biobjective assignment problem empirically follows a twodimentional normal distribution. Under this observation the supported nondominated points are on the lower left boundary of the bi-dimensional normal density and the non supported nondominated points are near the hypotenuse defined by two adajacent supported nondominated points. this explain why the final upper bounds in the exploration of the triangles were so small. This explains why the final upper bounds in the exploration of the triangles were so small and it also explains the gap between the two proposed methods. With the ranking approach we only enumerate solutions which are located near the boundary of the triangles, and therefore at the lower end of the distribution. Any slight increase in values encountered during enumeration can lead to a very large increase in the number of solutions enumerated.

Another consequence of this distribution is the number of efficient solutions. With a small range, the triangles defined by the supported nondominated points are often small, and because the other nondominated points are located near the hypotenuse of the triangle, there are fewer nonsupported nondominated points for a small range and more for a big range. However, the variance of objective values increases with increasing range of c_{ij}^k, so the number of equivalent solutions decreases. And since the upper bound is an integer the gap between the upper bound and the value of z^λ for the supported solutions remains the same, so the number of solutions inside this small band is larger. Consequently, the number of equivalent solutions with a very small range is very large and this kind of instance can be difficult to solve. In practice, with a range 20, the number of equivalent supported solutions is still significant.

7 Conclusions and Future Research

In this paper, we have presented a complete study of the two-phase method for the bi-objective assignment problem. We can conclude that the use of a ranking algorithm in Phase 2 implies a huge improvement of the performance of the method in comparison to the variable fixing strategy. This is validated by our experiments and by our observation of the distribution of objective values. We obtain a method which outperforms all existing methods

		instance	2AP05-1A20	2AP10-1A20	2AP15-1A20	2AP20-1A20	2AP25-1A20	2AP30-1A20	2AP35-1A20	2AP40-1A20
	#	X_{E_m}	8	16	39	55	74	88	81	127
		X_{SE_m}	3	6	12	13	25	27	27	54
		X_{NE_m}	5	10	27	42	49	61	54	73
		X_{E_M}	8	20	39	59	92	113	97	217
		X_{SE_M}	3	7	12	14	25	30	30	65
		X_{NE_M}	5	13	27	45	67	83	67	152
(1)	2ph	β_2	0	0.01	0.16	0.85	3.19	10.66	17.9	60.58
		β_1	0	0.02	0.19	0.93	3.43	12.58	20.72	116.36
	Seek & Cut β_2	$(PR1)(N1)$	0.8	1.77	5.96	11.16	10.96	19.8	27.16	45.51
		$(PR1)(N2)$	0.47	1.04	3.5	6.75	10.71	19.31	26.71	52.95
		$(PR2)(N1)$	0.86	2.15	7.36	12.89	14.65	26.21	33.99	57.56
		$(PR2)(N2)$	0.51	1.29	6.08	8.07	9.62	22.15	29.38	58.56
		$(PR3)(N1)$	1.31	4.19	21.94	44.92	36.5	87.57	90.46	130.04
	Seek & Cut β_1	$(PR1)(N1)$	0.78	1.72	5.92	11.26	11.18	21.24	29.29	84.21
		$(PR1)(N2)$	0.46	1.04	3.56	6.95	11.18	21.16	29.4	92.27
		$(PR2)(N1)$	0.8	2.12	7.37	12.9	14.89	28.09	36.61	96.84
		$(PR2)(N2)$	0.49	1.27	5.92	7.91	9.72	23.5	31.51	96.72
		$(PR3)(N1)$	1.26	4.14	21.88	44.93	36.70	89.29	92.94	168.48
		$(PR3)(N2)$	0.78	2.5	13.31	27.29	29.98	71.59	59.9	189.99
	rank-X_{NE_m}	β_1	0	0	0.13	0.51	2.05	5.32	11.21	63.72
		β_2	0	0	0.1	0.42	1.79	3.4	8.62	24.29
		β_1	0	0.01	0.06	0.19	0.46	1.01	1.8	4.71
		β_2	0	0.01	0.06	0.16	0.49	1.05	1.87	4.04
(2)	Cplex	CPLEX 9.0	0.02	0.14	0.72	1.71	3.13	5.63	7.02	12.21
		ranking β_2	0	0	0.02	0.07	0.18	0.39	0.73	1.55
(3)	(4)	# E	8	16	39	54	71	88	92	126
		CPU_t	5	10	14	61	102	183	384	1203

Table 1: (1) PowerPC G4 1 Ghz, 512 MB of RAM, under OS X system. (2) P4 EE 3,4 Ghz, RAM 4 GB, under Red Hat WS 3 Linux. (3) DEC 3000 alpha. (4) [20].

		instance	2AP45-1A20	2AP50-1A20	2AP60-1A20	2AP70-1A20	2AP80-1A20	2AP90-1A20	2AP100-1A20
	#	X_{E_m}	114	163	128	174	195	191	223
		X_{SE_m}	43	67	44	60	69	83	101
		X_{NE_m}	71	96	84	114	126	108	122
		X_{E_M}	170	406	234	413	694	947	1167
		X_{SE_M}	52	100	53	82	123	236	286
		X_{NE_M}	118	306	181	331	571	711	881
(1)	2ph	β_2	110.36	270.99	661.57	1980.15	3881.63	10181.73	14049.17
		β_1	170.48	703.09	1416.83	5726.72	13002.79	27916.14	61973.58
	Seek & Cut β_2	$(PR1)(N1)$	65.83	101.3	214.5	507.64	1057.23	1721.6	3188.85
		$(PR1)(N2)$	80.4	113.67	237.88	554.5	1078.08	1650.48	2730.15
		$(PR2)(N1)$	76.95	118.73	232.08	505.26	946.39	1676	2251.72
		$(PR2)(N2)$	100.98	187.35	366.64	639.35	1208.98	1590.71	2130.31
		$(PR3)(N1)$	152.57	259.74	325.95	647.44	1188.29	1803.19	2977.79
	Seek & Cut β_1	$(PR1)(N1)$	97.22	315.73	516.93	1951.02	4512.73	8395.28	31970.92
		$(PR1)(N2)$	114.87	331.40	539.33	2032.45	4516.04	7517.66	31565.47
		$(PR2)(N1)$	111.52	328.14	529.78	2029.92	4447.64	8439.2	31648.29
		$(PR2)(N2)$	133.3	392.77	619.34	2160.05	4537.33	7459.02	29330.84
		$(PR3)(N1)$	180.63	469.16	591.29	2064.43	4732.15	7740.68	31266.37
		$(PR3)(N2)$	180.76	627.5	690.96	2263.56	5543.77	8434.74	31446.25
	rank X_{NE_m}	β_1	68.48	267.1	362.83	1631.07	3857.59	6026.9	26815
		β_2	37.53	57.25	112.17	230.49	528	877.6	546.91
	rank	β_1	5.71	19.18	13.91	50.86	119.55	201.98	461.36
		β_2	5.4	13.56	11.63	40.32	76.99	120.53	228.26
(2)	Cplex	CPLEX 9.0	12.45	23.09	36.17	72.2	120.48	135.74	200.63
		ranking β_2	1.99	5.16	4.36	15.02	28.34	44.22	85.58
(3)	(4)	# E	113	156					
		CPU_t	3120	3622					

Table 2: (1) PowerPC G4 1 Ghz, 512 MB of RAM, under OS X system. (2) P4 EE 3,4 Ghz, RAM 4 GB, under Red Hat WS 3 Linux. (3) DEC 3000 alpha. (4) [20].

			size	10	20	30	40	50	60	70	80	90	100
Range 40		#	X_{E_m}	21	66	109	186	216	253	331	355	432	429
			X_{SE_m}	9	18	28	36	55	62	69	71	85	114
			X_{NE_m}	12	48	81	150	161	191	262	284	347	315
			X_{E_M}	21	68	119	195	324	323	437	458	711	666
			X_{SE_M}	9	18	32	37	62	72	75	74	92	123
			X_{NE_M}	12	50	87	158	262	251	362	384	619	543
	(1)	S & C (PR2)(N1) β_2		2.79	10.76	26.76	92.19	341.9	716.52	1788.91	2800.62	9131.49	17441.35
		rank	β_2	0.02	0.34	1.57	8.32	19.19	38.85	70.71	124.26	176.88	225.06
			β_1	0.02	0.34	1.58	8.39	19.61	41.43	76.79	129.74	210.19	284.31
	(2)	Cplex	CPLEX 9.0	0.2	3.01	10.91	43.88	68.63	146.13	230.5	328.16	481.5	512.96
			ranking β_2	0	0.13	0.64	3.44	7.26	14.78	26.92	46.55	66.36	83.63
Range 60		#	X_{E_m}	17	66	139	259	304	374	460	498	571	585
			X_{SE_m}	9	14	30	54	45	58	67	70	85	97
			X_{NE_m}	8	52	109	205	259	316	393	428	486	488
			X_{E_M}	17	66	148	279	341	425	522	591	696	685
			X_{SE_M}	9	14	31	57	47	59	69	70	87	103
			X_{NE_M}	8	52	117	222	294	366	453	521	609	582
	(1)	S & C (PR2)(N1) β_2		2.3	11.01	29.02	159.69	310.12	1269.92	3146.15	9563.08	20371.83	38553.18
		rank	β_2	0.02	0.49	2.6	10.3	23.44	52.43	100.98	175	351.31	399.65
			β_1	0.02	0.55	2.62	10.52	24.03	53.14	102.85	178.85	359.91	415.7
	(2)	Cplex	CPLEX 9.0	0.17	4.23	21.25	79.66	181.17	332.06	520.79	716.82	1172.82	1730.65
			ranking β_2	0	0.2	1	4.12	9.06	20.03	38.39	65.64	131.8	149.73

Table 3: (1) PowerPC G4 1 Ghz, 512 MB of RAM, under OS X system. (2) P4 EE 3,4 Ghz, RAM 4 GB, under Red Hat WS 3 Linux.

			size	10	20	30	40	50	60	70	80	90	100
Range 80		#	X_{E_m}	25	94	158	218	375	431	477	677	691	845
			X_{SE_m}	8	23	23	36	48	56	66	79	101	102
			X_{NE_m}	17	71	135	182	327	375	411	598	590	743
			X_{E_M}	25	95	161	250	396	458	513	773	759	1050
			X_{SE_M}	8	23	23	40	48	56	66	79	101	106
			X_{NE_M}	17	72	138	210	348	402	447	694	658	944
	(1)	S & C (PR2)(N1) β_2		3.11	12.28	31.70	151.11	528	1430.81	3644.27	13224.91	20411.17	53747.45
		rank	β_2	0.02	0.52	2.59	10.63	29.6	64.16	118.03	278.04	329.27	721.08
			β_1	0.02	0.54	2.67	10.67	29.74	64.88	120.33	282.33	338.74	774.85
	(2)	Cplex	CPLEX 9.0	0.24	5.45	34.31	79.33	271.04	630.10	851.87	1631.91	1918.06	3766
			ranking β_2	0	0.2	1.07	4.3	11.5	24.73	44.93	104.27	124.44	274.06
Range 100		#	X_{E_m}	13	82	169	243	301	470	573	671	722	947
			X_{SE_m}	7	17	25	35	33	55	69	77	83	85
			X_{NE_m}	6	65	144	208	268	415	504	594	639	859
			X_{E_M}	13	83	169	247	312	485	641	726	774	1046
			X_{SE_M}	7	17	25	36	34	55	73	77	84	89
			X_{NE_M}	6	66	144	211	278	430	568	649	690	957
	(1)	S & C (PR2)(N1) β_2		1.99	13.71	39	150.76	345.17	2455.2	4433.13	13209.98	26378.03	60227.31
		rank	β_2	0.01	0.25	3.52	10.37	21.68	98.76	144.37	257.59	370.09	711.97
			β_1	0.01	0.29	3.59	10.51	21.81	100.34	147/35	263.81	380.96	739.32
	(2)	Cplex	CPLEX 9.0	0.1	5.28	49.17	131.16	295.77	786.79	1252	1883.22	2732	4822
			ranking β_2	0	0.11	1.45	4.2	8.56	38.24	55.67	99.11	142.22	275.09

Table 4: (1) PowerPC G4 1 Ghz, 512 MB of RAM, under OS X system. (2) P4 EE 3,4 Ghz, RAM 4 GB, under Red Hat WS 3 Linux.

	Seek & Cut (PR2)(N1) β_2			Ranking β_2			Ranking β_1		
size	min	max	mean	min	max	mean	min	max	mean
10	1.52	2.93	2.26	0	0.02	0.01	0	0.02	0.01
20	6.75	15.52	10.33	0.08	0.25	0.16	0.08	0.26	0.18
30	20.75	29.96	23.88	0.80	1.91	1.10	0.89	2.02	1.19
40	45.28	69.74	57.15	2.23	5.73	3.40	2.46	6.98	4.44
50	92.89	155.05	138.70	5.57	11.14	9.22	5.90	12.53	10.50
60	262.68	379.11	304.76	14.83	32.71	21.68	18.07	49.19	28.34
70	394.94	749.68	521.76	25.36	46.4	34.76	34.71	58.13	46.40
80	627.78	1139.57	964.80	31.20	85.11	54.19	42.7	199.89	86.18
90	1201.10	2325.85	1614.93	54.32	112.36	85.18	85.32	158.23	125.15
100	1711.96	4049.85	2755.75	115.55	194.08	137.18	179.11	290.42	220.07

Table 5: CPU time in seconds for instances with range 20.

and which is very robust with respect to the range of objective coefficients. Our new goals are now to generalize the two-phase method to three objectives and more, and to extend our observations on the objective values to other kinds of instances.

References

[1] R.K. Ahuja, T.L. Maganti, and J.B. Orlin. *Network Flows - Theory, Algorithms, and Applications*. Prentice-Hall, 1993.

[2] C.R. Chegireddy and H.W. Hamacher. Algorithms for finding k-best perfect matchings. *Discrete Applieds Mathematics*, 18:155–165, 1987.

[3] F. Degoutin and X. Gandibleux. Un retour d'expérience sur la résolution de problèmes combinatoires bi-objectifs. 5e journée du groupe de travail Programmation Mathématique MultiObjectif (PM2O), Angers, France, 2002.

[4] M. Ehrgott. *Multicriteria Optimization*, volume 491 of *Lecture Notes in Economics and Mathematical Systems*. Springer Verlag, Berlin, 2000.

[5] M. Ehrgott and X. Gandibleux. Multiobjective combinatorial optimization. In M.Ehrgott and X. Gandibleux, eds., *Multiple Criteria Optimization: State of the Art Annotated Bibliographic Surveys*, volume 52 of *Kluwer's International Series in Operations Research and Management Science*, pages 369–444, 2002.

[6] M. Ehrgott and D.M. Ryan. The method of elastic constraints for multiobjective combinatorial optimization and its application in airline crew scheduling. In T. Tanino, T. Tanaka, and M. Inuiguchi, eds., *Multi-Objective Programming and Goal-Programming*, pages 117–122. Springer Verlag, Berlin, 2003.

[7] K. Fukuda and T. Matsui. Finding all the perfect matchings in bipartite graphs. *Networks*, 22:461–468, 1992.

[8] X. Gandibleux, H. Morita, and N. Katoh. Use of a genetic heritage for solving the assignment problem with two objectives. In C. Fonseca, P. Fleming, E. Zitzler, K. Deb, and L. Thiele, eds., *Evolutionnary Multi-Criterion Optimization*, volume 2632 of *Lecture Notes in Computer Sciences*, pages 43–57. Springer, 2003.

[9] X. Gandibleux, H. Morita, and N. Katoh. A population-based metaheuristic for solving assignment problems with two objectives. *Journal of Mathematical Modelling and Algorithms*, 2004. To appear.

[10] A.M. Geoffrion. Proper efficiency and the theory of vector maximization. *Journal of Mathematical Analysis and Applications*, 22:618–630, 1968.

[11] F. Glover and M. Laguna. *Tabu Search*. Kluwer Academic Publisher, Boston, 1997.

[12] H. W. Hamacher and M. Queyranne. K best solutions to combinatorial optimization problems. *Annals of Operations Research*, 4:123–143, 1985.

[13] P. Hansen. Bicriterion path problems. In G. Fandel and T. Gal, eds., *Multiple Criteria Decision Making Theory and Application*, volume 177 of *Lecture Notes in Economics and Mathematical Systems*, pages 109–127. Springer Verlag, Berlin, 1979.

[14] R. Malhotra, H.L. Bhatia, and M.C. Puri. Bi-criteria assignment problem. *Opsearch. Journal of the Operational Research Society of India*, 19(2):84–96, 1982.

[15] G. Mavrotas and D. Diakoulaki. A branch and bound algorithm for mixed zero-one multiple objective linear programming. *European Journal of Operational Research*, 107(3):530–541, 1998.

[16] J.B. Mazzola and A.W. Neebe. Resource-constrained assignment scheduling. *Operations Research*, 34(4):560–572, 1986.

[17] C.H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization*. Prentice Hall, Englewood Cliffs, 1982.

[18] P. Serafini. Some considerations about computational complexity for multiobjective combinatorial problems. In J. Jahn and W. Krabs, eds., *Recent Advances and Historical Development of Vector Optimization*, volume 294 of *Lecture notes in Economics and Mathematical Systems*. Springer Verlag, Berlin, 1986.

[19] R.E. Steuer. *Multiple Criteria Optimization: Theory, Computation and Application*. John Wiley & Sons, New York, NY, 1985.

[20] D. Tuyttens, J. Teghem, Ph. Fortemps, and K. Van Nieuwenhuyse. Performance of the mosa method for the bicriteria assignment problem. *Journal of Heuristics*, 6:295–310, 2000.

[21] E.L. Ulungu. *Optimisation Combinatoire multicritère: Détermination de l'ensemble des solutions efficaces et méthodes intéractives*. Doctoral thesis, Université de Mons-Hainault, Faculté des Sciences, 1993.

[22] E.L. Ulungu and J. Teghem. The two phases method: An efficient procedure to solve bi-objective combinatorial optimization problems. *Foundations of Computing and Decision Sciences*, 20:149–165, 1995.

[23] L.G. Valiant. The complexity of enumeration and reliability problems. *SIAM Journal on Computing*, 8:410–421, 1979.

Algorithm 1 procedure `BAP_Phase1`

Parameters $\downarrow$: The cost matrices C^1 and C^2
Parameters $\uparrow$: The set X_{SE_M}

--| Compute the lexicographically optimal solution x^1 for (z^1, z^2)
`solveAP` ($C^1 \downarrow,\ x^{1'} \uparrow,\ \bar{C} \uparrow$)
--| Consider (BAP$_\lambda$), the single objective optimisation problem with the cost matrix
--| $C^{1'} = \left[\lambda^1 c^1_{ij} + \lambda^2 c^2_{ij}\right]$ with $\lambda^1 = z^2(x^{1'}) + 1$ and $\lambda^2 = 1$
`solveAP` ($C^{1'} \downarrow,\ x^1 \uparrow,\ \bar{C} \uparrow$)

--| Compute the lexicographically optimal solution x^2 for (z^2, z^1)
`solveAP` ($C^2 \downarrow,\ x^{2'} \uparrow,\ \bar{C} \uparrow$)
--| Consider (BAP$_\lambda$), the single objective optimisation problem with the cost matrix
--| $C^{2'} = \left[\lambda^1 c^1_{ij} + \lambda^2 c^2_{ij}\right]$ with $\lambda^1 = 1$ and $\lambda^2 = z^1(x^{2'}) + 1$
`solveAP` ($C^{2'} \downarrow,\ x^2 \uparrow,\ \bar{C} \uparrow$)

--| Compute X_{SE_M}
$S \leftarrow \{\ x^1,\ x^2\ \}$; `solveRecursion`($x^1 \downarrow,\ x^2 \downarrow,\ S \updownarrow$); $X_{SE_M} \leftarrow S$

Comment. In the algorithms, the symbols $\downarrow$, $\uparrow$ and $\updownarrow$ specify the transmission mode of a parameter to a procedure; they correspond respectively to the mode IN, OUT and IN OUT. The symbol --| marks the beginning of a comment line.

Algorithm 2 procedure `solveRecursion`

Parameters $\downarrow$: x^r, x^s
Parameters $\updownarrow$: S

--| Compute the set R of optimal solutions x of (BAP$_\lambda$):
--| $\min\{\lambda^1 z^1(x) + \lambda^2 z^2(x) : x \in X\}$
--| where $\lambda^1 = z^2(x^r) - z^2(x^s)$, and $\lambda^2 = z^1(x^s) - z^1(x^r)$.
--| $c^\lambda_{ij} = \lambda^1 c^1_{ij} + \lambda^2 c^2_{ij}$, $i = 1, \ldots, n$; $j = 1, \ldots, n$
`solveAP` ($C^\lambda \downarrow$, $x \uparrow$, $\bar{C}^\lambda \uparrow$); `enumerate` ($\bar{C}^\lambda \downarrow$, $x \downarrow$, $R \uparrow$); $R \leftarrow R \cup \{x\}$
$S \leftarrow S \cup R$

if $\{z^\lambda(x) : x \in R\} \cap \overline{z(x^r)z(x^s)} = \emptyset$ **then**
--| Case a)
--| Let x^{t_1} and x^{t_2}, the solutions of R with respectively the min. and max. value on z^1
`solveRecursion` ($x^r \downarrow$, $x^{t_1} \downarrow$, $S \updownarrow$); `solveRecursion` ($x^{t_2} \downarrow$, $x^s \downarrow$, $S \updownarrow$)
end if

--| if $\{z^\lambda(x) : x \in R\} \subset \overline{z(x^r)z(x^s)}$ then
--| Case b) : nothing to do

Algorithm 3 procedure `BAP_Phase2`

Parameters $\downarrow$: C^1, C^2, X_{SE_m}
Parameters $\uparrow$: S which is X_{NE} or X_{NE_M} depending on the upper bound (β_1 or β_2) selected

```
S ← ∅
for all x^r, x^s adjacent in X_{SE_m} do
  --| Compute one optimal solution x̃ of (BAP_λ) with the cost matrix
  --|   C^λ = [λ^1 c^1_ij + λ^2 c^2_ij] with λ^1 = z^2(x^r) − z^2(x^s) and λ^2 = z^1(x^s) − z^1(x^r)
  solveAP ( C^λ ↓, x̃ ↑, C̄^λ ↑ )

  --| Let C̄^λ be a reduced cost matrix of z^λ for the problem (BAP_λ)
  --|      C̄^1 be a reduced cost matrix of z^1 for the solution x^r
  --|      C̄^2 be a reduced cost matrix of z^2 for the solution x^s
  --|      X_PE be the list of potentially efficient solutions of the triangle
  --|      vUB be the initial value for the upper bound

  L ← {(i,j) | c̄^λ_ij > 0}
  X_PE ← ...                                  --| Empty or initialised with a heuristic (Section 5.3)
  if X_PE ≠ ∅ then
    computeUpperBound(X_PE ↓, vUB ↕)
  else
    vUB ← z^λ(x^r) + λ^1 λ^2 − λ^1 − λ^2
  end if

  if vUB > z^λ(x^r) then

    --| Test if the assignments give a solution outside of the triangle
    for all (i,j) ∈ L do
      if not( isTest_123( C̄^λ ↓, C̄^1 ↓, C̄^2 ↓, x^r ↓, x^s ↓, (i,j) ↓) ) then
        L ← L \ {(i,j)}
      end if
    end for

    if L ≠ ∅ then
      --| begin to impose assignments
      L ← L ∪ {(i,j) | c̄^λ_ij = 0}
      for all (i,j) ∈ L do
        if NeedtoFixVariable( C̄^λ ↓, C̄^1 ↓, C̄^2 ↓, x^r ↓, x^s ↓, (i,j) ↓) then
          impose( {(i,j)} ↓, L ↓, vUB ↕, X_PE ↕)
        end if
        C̄^λ(x_i) ← ∞; C̄^1(x_i) ← ∞; C̄^2(x_i) ← ∞; C^1(x_i) ← ∞; C^2(x_i) ← ∞
      end for
    end if
  end if
  S ← S ∪ X_PE
end for
```

Algorithm 4 procedure `impose`

Parameters $\downarrow$: $LIST$, L
Parameters $\updownarrow$: vUB, X_{PE}

--| Let $C^{\lambda}r$ be the costs matrix of (BAP_{λ}) after the new imposition
--| Let $\bar{C}^{\lambda}r,\ \bar{C}^{1}r,\ \bar{C}^{2}r$ be reduced costs matrix after the new imposition
--| Let $LIST$ be the assignments to impose
`solveAP`($C^{\lambda}r\downarrow,\ x\uparrow,\ \bar{C}^{\lambda}r\uparrow$)
if ($z^{\lambda}(x) \leq vUB$) **then**
  if `isInTriangle`($x^{r}\downarrow,\ x^{s}\downarrow,\ x\downarrow$) **and not**(`isDominated`($x\downarrow,\ X_{PE}\downarrow$)) **then**
    $X_{PE} \leftarrow X_{PE} \cup \{x\}$
    `removeDominatedSolution`($X_{PE}\updownarrow$)
    `computeUpperBound`($X_{PE}\downarrow,\ vUB\updownarrow$)
  end if

  --| Let i be the index of the last assignment (pair) in $LIST$
  for all $j > i$ according to L **do**
    if `compatible`($x_j\downarrow,\ LIST\downarrow$) **then**
      if `NeedtoFixVariable`($\bar{C}^{\lambda}r\downarrow,\ \bar{C}^{1}r\downarrow,\ \bar{C}^{2}r\downarrow,\ x\downarrow,\ x_j\downarrow$) **then**
        $LIST \leftarrow LIST \cup \{x_j\}$
        `impose`($LIST\downarrow,\ L\downarrow,\ vUB\updownarrow,\ X_{PE}\updownarrow$)
        $LIST \leftarrow LIST \setminus \{x_j\}$
      end if
      - -| for the following assignments, it is necessary to place x_j at its place in the reduced problem
      $\bar{C}^{\lambda}r(x_j) \leftarrow \infty$; $\bar{C}^{1}r(x_j) \leftarrow \infty$,;$\bar{C}^{2}r(x_j) \leftarrow \infty$; $C^{1}r(x_j) \leftarrow \infty$; $C^{2}r(x_j) \leftarrow \infty$
    end if
  end for
end if

Algorithm 5 procedure `BAP_Phase2_ranking`

Parameters $\downarrow$: C^1, C^2, X_{SE_m}
Parameters $\uparrow$: S which is X_{NE} or X_{NE_M} according to the upper bound (β_1 or β_2) selected

```
S ← ∅
for all x^r, x^s adjacent in X_{SE_m} do
  --| Compute one optimal solution x̃ of (BAP_λ) with the cost matrix
  --|    C^λ = [λ^1 c^1_ij + λ^2 c^2_ij] with λ^1 = z^2(x^r) − z^2(x^s) and λ^2 = z^1(x^s) − z^1(x^r)
  solveAP ( C^λ ↓, x̃ ↑, C̄^λ ↑ )

  --| Let C̄^λ be a reduced cost matrix of z^λ for the problem (BAP_λ)
  --|      C̄^1 be a reduced cost matrix of z^1 for the solution x^r
  --|      C̄^2 be a reduced cost matrix of z^2 for the solution x^s

  L ← {(i,j) : c̄^λ_ij > 0}
  R ← ∅                                          --| R, the list of efficient solutions in the triangle
  vUB ← z^λ(x^r) + λ^1 λ^2 − λ^1 − λ^2           --| vUB, the initial value for the upper bound
  if vUB > z^λ(x^r) then

    --| Test if the assignments give a solution outside of the triangle
    for all (i,j) ∈ L do
      if not( isTest_123( C̄^λ ↓, C̄^1 ↓, C̄^2 ↓, x^r ↓, x^s ↓, (i,j) ↓) ) then
        c^λ_ij ← ∞
      end if
    end for

    if L ≠ ∅ then
      --| begin the ranking
      K ← 0
      while (z^λ(x^K) ≤ vUB) do
        K ← K + 1
        ComputeNextK_bestSolution(K ↓, C^λ ↓, x^K ↑)
        if isInTriangle(x^r ↓, x^s ↓, x^K ↓) and not( isDominated(x^K ↓, R ↓) ) then
          R ← R ∪ {x^K}
          computeUpperBound(R ↓, vUB ↕)          --| according to β_1 or β_2
        end if
      end while
    end if
  end if
  S ← S ∪ R
end for
```

The Biobjective Assignment Problem

Anthony Przybylski, Xavier Gandibleux, Matthias Ehrgott

Abstract

In this paper we present a synthesis of the two phase method for the biobjective assignment problem. The method, which is a general technique to solve multiobjective combinatorial optimization (MOCO) problems, has been introduced by Ulungu in 1993. However, no complete description of the method to find all efficient solutions of the biobjective assignment problem (that allows an independent implementation) has been published.
First, we present a complete description of a two phase method for the biobjective assignment problem, with an improved upper bound. Second, a combination of this method with a population based heuristic using path relinking is proposed to improve computational performance. Third, we propose a new technique for the second phase with a ranking approach.
All of the methods have been tested on instances of varying size and range of objective function coefficients. We discuss the obtained results and explain our observations based on the distribution of objective function values.

Additional Key Words and Phrases: Multiobjective combinatorial optimization, assignment problem, exact algorithm, population based heuristic, path-relinking, efficient solutions, k-best solution, ranking

LINA, Université de Nantes
2, rue de la Houssinière
B.P. 92208 — F-44322 NANTES CEDEX 3