Laboratoire d'Automatique, de Mécanique et d'Informatique industrielles et Humaines — UMR 8530 CNRS

Recherche Opérationnelle et Informatique

Note de Recherche

n° 7 / 2003 / LAMIH-ROI

Population based-heuristic for solving the assignment problems with two objectives

Xavier Gandibleux, Hiroyuki Morita, Naoki Katoh

Submitted October 12, 2003
to Journal of Mathematical Modelling and Algorithms

UNIVERSITÉ DE VALENCIENNES ET DU HAINAUT-CAMBRÉSIS
LE MONT HOUY — F-59313 VALENCIENNES CEDEX 9 — FRANCE
Secrétariat : Mme Aureggi – phone: +33 (0)3 27 51 19 41
www.univ-valenciennes.fr/LAMIH

A population-based metaheuristic for solving assignment problems with two objectives

Xavier Gandibleux (xavier.gandibleux@univ-valenciennes.fr)
LAMIH/ROI – UMR CNRS 8530, Université de Valenciennes F-59313 Valenciennes cedex 9, France

Hiroyuki Morita (morita@eco.osakafu-u.ac.jp)
Faculty of Economics, Osaka Prefecture University Sakai, Osaka 599-8231, Japan

Naoki Katoh (naoki@archi.kyoto-u.ac.jp)
Graduate School of Engineering, Kyoto University Kyoto 606-8501, Japan

October 12, 2003

Abstract. The paper presents a population-based heuristic for computing approximations of the efficient solution set for the assignment problem with two objectives. The heuristic is based on the intensive use of three operators - a local search, a crossover and a path-relinking - performed on a population composed only of elite solutions. The initial population is a subset of supported solutions, where each solution is one optimal assignment for an appropriate weighted sum of two objectives. Genetic information is derived from the elite solutions, providing a useful genetic heritage to be exploited by crossover operators. An upper bound set, defined in the objective space, provides one acceptable limit for performing a local search. Results of extensive numerical experiments have shown that the heuristic is able to quickly find a very good approximation of the efficient frontier, and outperforms all previously published results. In addition, this heuristic has two main advantages. It is based on simple easy-to-implement principles, and it does not need a parameter tuning phase.

Keywords: multiobjective combinatorial optimization, assignment problem, multiobjective heuristic

1. Introduction

1.1. The assignment problem with two objectives

The assignment problem (AP) is a well-known fundamental combinatorial optimization problem. Several practical situations can be formulated as an AP. In addition, AP is a sub-problem of more complicated ones, including transportation problems, distribution problems and traveling salesman problems. Efficient specific algorithms exist to solve AP involving a single objective, such as the Hungarian method or the successive shortest path method.

In a multiple objective framework, the assignment problem with two objectives ($biAP$) can be formulated as follows, where c_{il}^q are non-negative integers and $x = (x_{11}, \ldots, x_{nn})$:

$$\left[\begin{array}{lll} \text{“min”} z^q(x) = & \displaystyle\sum_{i=1}^{n}\sum_{l=1}^{n} c_{il}^q x_{il} & q = 1,2 \\ & \displaystyle\sum_{i=1}^{n} x_{il} = 1 & l = 1,\ldots,n \\ & \displaystyle\sum_{l=1}^{n} x_{il} = 1 & i = 1,\ldots,n \\ & x_{il} \in \{0,1\} & \end{array}\right] \qquad (biAP)$$

Let $x, x^\star \in X$ be two feasible solutions of a problem ($biAP$), where X denotes *the decision space*, and $z^1(x), z^2(x)$ denote two objective functions defined on X. Let $z(x) = (z^1(x), z^2(x))$ be the objective vector associated with the decision variable x, and let $Z = \{z(x) \mid x \in X\}$ be *the objective space*. An objective vector $z(x)$ is called *a point* of Z. A solution $x^\star$ is *efficient* if no other feasible solution x exists, such that $z^q(x) \leq z^q(x^\star),\ q = 1,2$ with at least one strict inequality. $z(x^\star)$ is *a non-dominated point* in the objective space Z. The set of efficient solutions is denoted by E, and the representation of E in Z is called the *efficient frontier*.

The set of *supported efficient solutions*, denoted by SE, is a subset of E, such that $z(x)$ for $x \in SE$ is on the lower-left boundary of the convex hull for $\{z(x) : x \in X\}$. For $\lambda = (\lambda_1, \lambda_2)$ with $\sum_{q=1}^{2} \lambda_q = 1$ and $\lambda_q > 0$, let ($biAP_\lambda$) denote the parametrized single objective problem: $\min\{\sum_{q=1}^{2} \lambda_q z^q(x) : x \in X\}$. The set SE is composed of $\{x \in X$ such that x is optimal to ($biAP_\lambda$) for some $\lambda = (\lambda_1, \lambda_2)\}$. The computation of each ($biAP_\lambda$) for a fixed λ can be implemented easily by relaxing the integrality constraints and solving the problem using a parametric simplex method, or any specific solving algorithm. Consequently, E is partitioned into two subsets, the set SE and the set $NE = E \setminus SE$ of *non-supported efficient solutions*.

1.2. Classification of efficient solutions

In general, the set E is not composed of a single solution. The size of E is usually large, and efficient solutions can be characterized and classified into several different subsets.

- The *complete set* (E) and the *minimal complete set* (E_m) of efficient solutions. Generally, several distinct efficient solutions $x_1^*, x_2^*, x_3^*, \ldots$ can correspond to the same non-dominated point $z(x_1^*) =$

$z(x_2^*) = z(x_3^*) = \cdots$ in the objective space. The solutions $x_1^*, x_2^*, x_3^*, ...$ are said to be *equivalent* in the objective space. The number of such equivalent solutions is important; the enumeration of all of them may be intractable. This is the case for some instances of the biobjective shortest path problem in which all feasible solutions are efficient (Hansen, 1979). In this situation, it is impossible to design an efficient algorithm for computing the *complete set* of efficient solutions (i.e. all feasible solutions of X which correspond to a non-dominated point in the objective space Z).

Hansen (1979) introduced the *minimal complete set* E_m of efficient solutions (i.e. the subset of E that contains no equivalent solutions, and for any $x \in E$ there exists $x' \in E_m$ such that x and x' are equivalent.) This restricted efficient set opens the possibility of designing a polynomial algorithm.

- The *supported* (SE) and the *non-supported* (NE) solutions. If a convex hull is computed in the objective space for all solutions in E, then the supported solutions belong to the boundary of this convex hull. Any supported solution can be computed by solving the ($biAP_\lambda$) for some λ. The non-supported efficient solutions are the efficient solutions not located on the boundary of the convex hull. In the biobjective case, NE solutions are located in the triangles drawn on two successive supported efficient solutions in the objective space. There is no theoretical characterisation which leads to the efficient computation of NE solutions. Applying the minimal complete set definition to SE and NE gives rise to the SE_m and NE_m sets.

- The supported solutions *located on the vertex* ($SE1$) and those *not located on the vertex* ($SE2$) of the convex hull. Both solution sets can be obtained by solving the ($biAP_\lambda$). Nevertheless, computing the $SE2$ set is generally more difficult that computing $SE1$, because the former requires the enumeration of all optimal solutions, which minimizes $\lambda z^1(x) + (1 - \lambda)z^2(x)$ with λ given. This enumeration is not trivial in general. Obviously, applying the minimal complete set definition to $SE1$ and $SE2$ gives rise to the $SE1_m$ and $SE2_m$ sets.

Figure 1 summarizes the inclusion relationship among the sets defined above, illustrating that, for example, $SE1_m \subseteq SE_m \subseteq SE$. The papers published are sometimes unclear concerning the ability of discussed algorithms. Some authors in the literature claim that their algorithm can enumerate "all" efficient solutions in terms of the set E. However, it is generally difficult to compute this set as mentioned

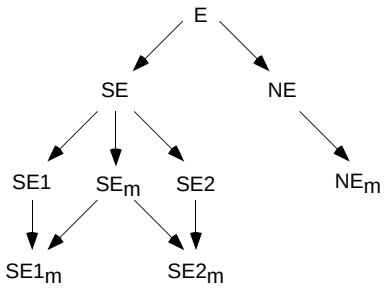

Figure 1. Classification of efficient solutions.

before. Then, it is important to highlight the class of efficient solutions handled by the algorithm.

1.3. Literature

According to Ehrgott and Gandibleux (2002), the papers at an earlier stage concerning multiple objective AP only deal with SE, using convex combinations of objectives, or goal programming. Exact algorithms to determine the whole set E have been proposed (Malhotra et al., 1982) (Ulungu, 1993). These algorithms make use of single objective methods and the duality properties of the assignment problem. However, exact methods for ($biAP$) are time consuming (Tuyttens et al., 2000). Consequently approximation methods are needed to obtain a compromise between solution quality and reasonable computing time. The introduction of metaheuristic techniques for solving multiobjective programming problems has mushroomed over the last ten years. This activity has given birth to multiobjective metaheuristics (MOMH), aimed at approximating the (sub)set of efficient solutions (see Gandibleux et al. -in press- for a view of recent developments in this area).

An extension of simulated annealing to deal with multiple objectives, called the MOSA method (Ulungu, 1993), has also been applied to the assignment problem. The results reported in Tuyttens et al. (2000) for an improved version of MOSA show the limit of such a general purpose meta-heuristic; MOSA quickly encounters difficulties in producing good approximations for ($biAP$). It is able to find some exact efficient solutions only for small instance size (up to 30×30 variables).

For the knapsack problem and a scheduling problem with two objectives, the advantage of using good genetic information in a MOMH has been underlined in our previous works (Morita et al., 2001), (Gandibleux et al., 2001). Such "genetic information" can be deduced from the set (or subset) of supported efficient solutions, or from an approximation

of the efficient set given by a heuristic. Recently, we presented a first study using genetic information for solving the ($biAP$) (Gandibleux et al., 2003a 2003b). Our results, compared to those produced using MOSA, have demonstrated the interest of this approach. However, the algorithm has shown some limitations during its progression in the detection of potential efficient solutions.

In this paper, we present an improved efficient heuristic for assignment problems with two objectives. This method is influenced by several key factors related to multiobjective metaheuristic and multiobjective combinatorial optimisation, and it falls into the category of evolutionary multiobjective optimization methods. The main principles and an algorithmic description of the heuristic are presented in section 2. The numerical results are reported and compared with previously published results in section 3. The last section gives a conclusion and some directions for future developments.

2. The principles of the heuristic and its algorithmic description

Two numerical instances are used throughout this section to illustrate the main principles of the heuristic. The first instance involves 5×5 variables (denoted `2AP5-1A20.dat` in section 3). The assignment costs for the objectives are :

$$c^1 = \begin{bmatrix} 13 & 14 & 7 & 2 & 11 \\ 5 & 10 & 11 & 7 & 10 \\ 7 & 19 & 9 & 16 & 19 \\ 3 & 19 & 10 & 0 & 6 \\ 12 & 9 & 2 & 4 & 15 \end{bmatrix} \quad c^2 = \begin{bmatrix} 1 & 13 & 15 & 18 & 3 \\ 0 & 3 & 17 & 8 & 6 \\ 9 & 5 & 8 & 0 & 4 \\ 18 & 19 & 3 & 19 & 7 \\ 2 & 3 & 19 & 12 & 15 \end{bmatrix}$$

This small instance is used as a didactic example. The second instance involves 100×100 variables (denoted `2AP100-1A20.dat` in section 3). This large instance is used to exhibit the behavior of the analyzed component in the objective space.

2.1. General principle

The principle of the heuristic is based on the intensive use of three operators performed on a population composed of only elite solutions. Here we use the terminology, "elite solution", to indicate a solution in the population such that no other solution dominates that solution. Such a solution is also called *a potential efficient solution*. The sets of these solutions are respectively named "elite solution set" and "potential solution set (denoted by PE)".

The population is first initialized with a subset of supported solutions (routine `detectPEinit`). The three operators are a local search (routine `localSearchOverNewElites`), a crossover (routine `crossover WithElites`) and a path-relinking (routine `pathRelinkingWithElites`). An upper bound set defined in the objective space (routine `buildUpper BoundSet`) provides an acceptable limit for performing a local search. A genetic map is derived from the elite solutions (routine `elaborate GeneticInformation`). This map provides useful information for fixing certain bits by crossover operators. This genetic information is refreshed periodically. Each new potential solution is noted, so that the heuristic stops when no new potential efficient solutions have been produced after a series of iterations (routine `noteNewSolutions`). The heuristic can also be stopped after a predefined effort (parameter `iterationMax`), possibly combined with the detection of unfruitful iterations (routine `isTheEnd?`).

The main part of the heuristic is described by Algorithm 1. Each iteration of the algorithm performs one crossover operation and one path-relinking operation, which generates respectively one solution, and a list of solutions. For each of these generated solutions, a local search is then performed if and only if the solution is located in the "admissible area". All solutions in this neighborhood which are potentially efficient are added to the current set PE. The iteration is finished in performing again a local search operation for each potential efficient solution newly included in PE set.

Due to the simplicity of the method, performing one iteration consumes much less CPU time than other conventional population-based heuristics, such as genetic algorithms, especially when the individual is not promising (in this case, no local search is performed). Consequently, the heuristic can be very aggressive in implementing a generation process that performs many iterations. The approximation set contains only potential efficient solutions. The algorithm maintains PE, and iteratively improves PE towards E (the set of exact efficient solutions). Thus, a poor solution, one that is far from the exact efficient frontier, will never be introduced in the approximation. In any time, the heuristic reports only good approximation of the efficient frontier.

In contrast with other MultiObjective Evolutionary Algorithms (Coello et al., 2002), our heuristic performs no direction searches to drive the approximation process, and it does not require any ranking method (there is no fitness measure). This is a remarkable fact, given that direction searches and ranking are often criticized; the former for guiding the heuristic search, and the latter for provoking increased computing effort. The following subsections describe the main aspects of the heuristic in more detail.

2.2. The initial population

$SE1_m$, the minimal complete set of solutions located on the vertex of the convex hull, is computed by solving a series of $(biAP_\lambda)$, the parametric problem. The resolution principle follows a dichotomic scheme (algorithms 2 and 3). The single objective assignment problem is solved using the *Successive Shortest Path* algorithm (routine `succShortestPath`). Details concerning this algorithm are available in Ahuja et al. (1993).

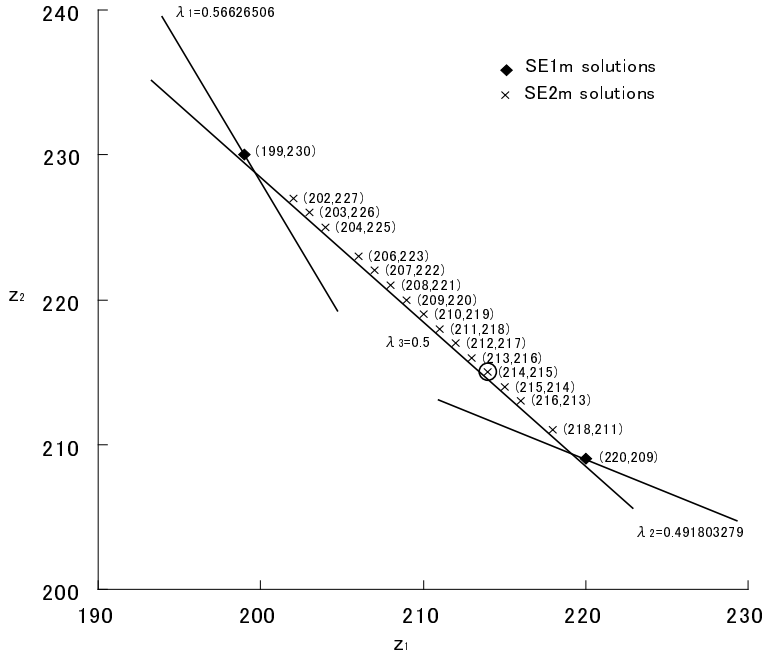

Figure 2. Example (instance `2AP100-1A20.dat`) where the optimal solution $\hat{x}$ for $(biAP_\lambda)$ with $\lambda = 0.5$ points out $z(\hat{x}) = (214, 215)$, a solution in $SE2_m$

It is important to notice that several efficient solutions can be obtained from $SE2_m$ set using this resolution principle, as illustrated in figure 2 (the encircled solution). Obviously, these solutions are integrated into the initial elite population set, denoted by PE_{init}. However, no specific procedure has been developed for computing solutions belonging to $SE2_m$. The initial population for the didactic problem is reported in table I. These three solutions belong only to $SE1_m$.

In another context, the SE_m was computed with Cplex for all instances used in section 3. This information has been used for evaluating the heuristic's detection ratio.

The section concerning numerical experiments reports an analysis where the heuristic uses first SE_m, and second PE_{init}, in order to evaluate the impact of using a poorer initial elite set.

Table I. The didactic example has three solutions in the $SE1_m$ set

SE	x_0	x_1	x_2	x_3	x_4	z^1	z^2
1	4	2	1	5	3	27	56
2	5	1	4	3	2	51	9
3	4	1	3	5	2	31	36

2.3. The upper bound set

The upper bound set is defined as the set of "local nadir points", where one nadir point is derived from two adjacent supported solutions. More precisely, if x_1 and x_2 are two adjacent supported solutions in the objective space, the corresponding nadir is a point in Z with the following coordinates :

$$\left(\max\left(z^1(x_1), z^1(x_2)\right), \max\left(z^2(x_1), z^2(x_2)\right)\right)$$

Figure 3 illustrates the upper bound set for the didactic example.

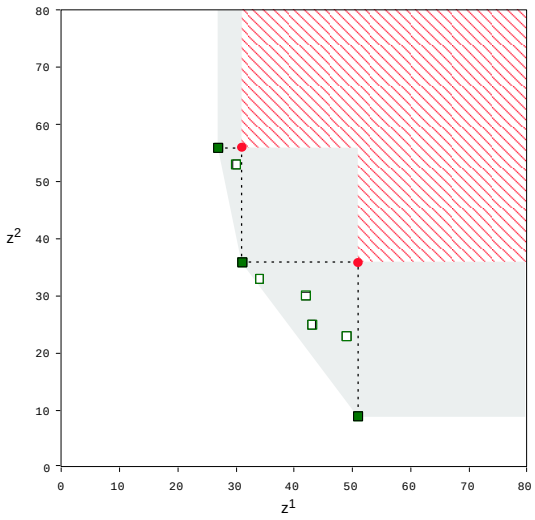

Figure 3. Efficient solutions (squares), upper bound set (bullets) and the admissible area (grey) where a local search procedure will be performed on new solutions. Triangles with dashed lines denote areas where efficient solutions can exist.

The upper bound set is used in a heuristic strategy to decide whether or not a solution is a candidate for an intensive search in its neighborhood. This strategy helps to save computing effort; when a solution is not dominated by the nadirs belonging to the bound set, it is located in the admissible area. Then this solution is checked to see if it is a new potential efficient solution. All solutions in this area are considered to

be promising for finding new potential efficient solutions. A local search is then performed starting from such promising solution. Otherwise no effort is spent on solutions outside of the admissible area.

2.4. The local search operator

An assignment is coded as a permutation of tasks $(i_1, i_2, ..., i_n)$ in position $(1, 2, ..., n)$. For example, the coded solution $x = (4, 2, 1, 5, 3)$ means task $i = 4$ is assigned to position $j = 1$, $i = 2$ to $j = 2$, etc.

We adopted a classic neighborhood structure. Starting from a solution x, we consider the swap move (i_{j1}, i_{j2}) between two tasks. The set of pairwise exchanges (i_{j1}, i_{j2}), with $j1 = 1, ..., n-1$ and $j2 = j1+1, ..., n$, defines an associated neighborhood $\mathcal{N}(x)$ for the current solution x. The local search is described by Algorithm 4. Because the computational cost for a local search can be significant ($O(n^2)$), the local search is triggered only for promising candidate solutions resulting from the crossover and the path-relinking operators.

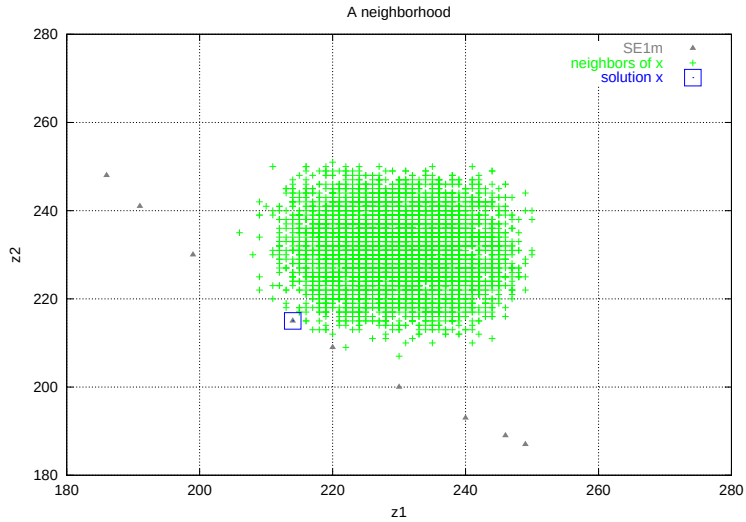

Figure 4. The neighbors of one solution $x \in SE1m$ for the `2AP100-1A20.dat` instance

The local search routine is also called in `localSearchOverNewElites` operator, which ensures that a local search is performed only once for each solution in the elite set (algorithm 5).

2.5. The genetic map

The mechanism used here is inspired by the principle of pheromones in an artificial ant colony (Dorigo and Di Caro, 1999). Assuming that

similarities exist among assignment vectors of efficient solutions, the occurrence frequency of assignment "i in position j" in vector x for elite solutions is computed and summarized in a roulette wheel that provides the genetic information. A roulette wheel is elaborated for each position, and the genetic map is made up of roulette wheels. The map is the genetic heritage of our population, and this information is used intensively by the crossover operator (subsection 2.6).

The genetic information is always derived from elite solutions. The initial map is then derived only from exact supported solutions. For the didactic example, the elite set contains 3 solutions (table I). The assignments observed in position 1 (first wheel) are the task 4 twice and the task 5 once. Given a uniform roulette wheel principle, the probability of selection the task i in position $j = 1$ is then (0 , 0 , 0 , 2/3 , 1/3). This information is summarized on the first wheel of the genetic map shown in figure 5.

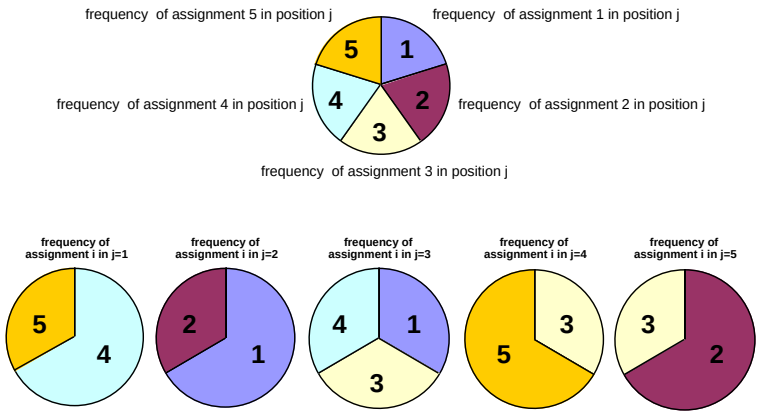

Figure 5. Occurrence frequency of task i in position j for the set PE.

The genetic map is refreshed periodically. Roulette wheels are rebuilt using PE, the current set of elite solutions, when PE has been significantly renewed. In the current version, refreshment occurs during generation, after a predefined number of iterations (parameter `refreshMapFrequency`) has been performed. The parameter value is experimentally set at 100 000 iterations.

2.6. The crossover operator

For each crossover operation, two parent individuals are randomly selected from the current elite population, and one offspring is produced. Genes common to the parents are copied in the child. The other genes are determined using the genetic map, on the basis of the occurrence frequencies stored in the roulette wheel. To illustrate this second step,

consider $I1$ and $I2$, two individuals in the didactic example, with $I3$ being the solution resulting from the crossover :

j			1	2	3	4	5	
$I1$	=	(	4	2	1	5	3	)
$I2$	=	(	4	1	3	5	2	)
$I3$	=	(	4	*	*	5	*	)

The function `useGeneticInformation( j , map )` is called using $j = 2, 3, 5$ and the genetic map given in figure 5. The three genes are then fixed as follows:

1. To determine a task i in position $j = 2$, the map's 2nd wheel suggests to choose one from the set $\{1, 2\}$ with a selection probability of $2/3$ and $1/3$ respectively. Using this information, the task is randomly selected according to the roulette principle. In this way, task 2, for example, is selected (function `selectAssignmentUsing RouletteWheel`).

2. For the position $j = 3$, the 3rd wheel suggests tasks $\{1, 3, 4\}$. However, assigning task 4 to position 3 produces an infeasible solution. Consequently, the wheel is reduced, so that it contains only feasible assignments for the position in question, in this case, tasks $\{1, 3\}$. Task 3, for example, is selected according to the roulette principle (function `selectAssignmentUsingRouletteWheel`).

3. In position $j = 5$, the wheel suggests tasks $\{2, 3\}$. But none of them is a feasible choice in this position. Therefore, the wheel is emptied, since no feasible choice exists. A list of feasible tasks (LFT) is then elaborated for this position, and one task is randomly selected. In this case, LFT contains only task $\{1\}$. Thus, the random selection of a task from the LFT results in the assignment of task 1 to position 5 (function `selectFeasibleAssignmentRandomly`).

Finally, the bound set is used to check that the solution produced $I3$ falls in the admissible area. If so, the solution is compared with the current list of potential solutions, and a local search is performed. Otherwise, the improvement strategy is not triggered, and the solution is simply ignored. Figure 6 shows a sample of the solutions produced by this operator during the course of the algorithm for the `2AP100-1A20.dat` instance. The main steps of this operator are summarized in Algorithms 6 and 7.

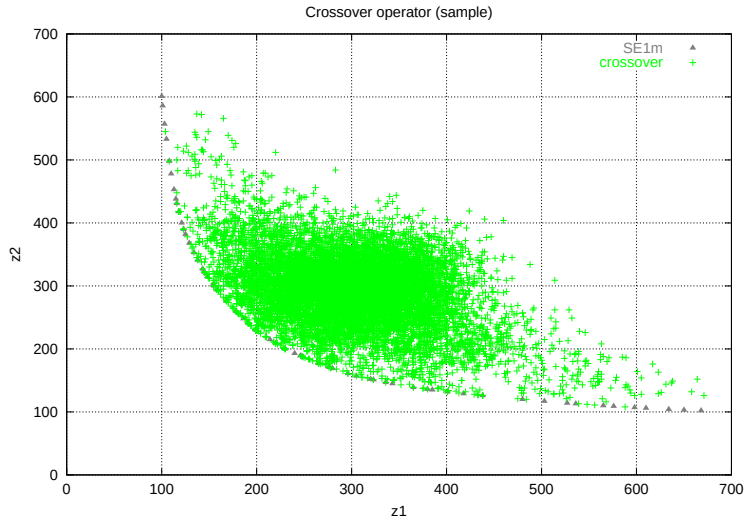

Figure 6. Sample of the solutions produced by the crossover operator for the `2AP100-1A20.dat` instance

2.7. The path-relinking operator

Path-relinking generates new solutions by exploring the trajectories that connect elite solutions, starting from one solution -the initiating solution-, and generating a path through the neighborhood space that leads toward the other solution -the guiding solution- (Glover and Laguna, 1997). Because the population contains elite solutions, the presence of a path-relinking operator in our heuristic is a natural development.

A path-relinking operation starts by randomly selecting $I1$ and $I2$, two individuals in the current elite population. Because both are elites, both could potentially be the guiding solution. Let $I1$ be the initiating solution and $I2$ the guiding solution. The path-relinking operation generates a path $I1(= I_0), I_1, \ldots, I2$, such that the distance between I_i and $I2$ decreases in i, where distance is defined as the number of different positions for tasks assigned in I_i and $I2$.

A path is generated as follows: Let $i = 0$. The list LDA reports the tasks assigned to different position in solutions I_i and $I2$. A new solution is built by randomly selecting one task from LDA, and assigning this task to its final position j in I_i, which is its position in the guiding solution $I2$. In order 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. Let us illustrate the principle using the didactic example :

j			1	2	3	4	5	
$I1$	=	(	[4]	2	1	[5]	3	)
$I2$	=	(	[4]	1	3	[5]	2	)
I_0	=	(	[4]	2	1	[5]	3	)
LDA	=	(	1	3	2	)	→	3
I_1	=	(	[4]	2	**3**	[5]	**1**	)
LDA	=	(	1	2	)		→	1
I_2	=	(	[4]	**1**	[3]	[5]	**2**	)

As shown above, only one solution is built in the neighborhood of I_i at each iteration, which limits the effort needed, in avoiding the examination of the other candidates in the neighborhood. The fact that the task is randomly selected from the list LDA introduces a form of diversity in the solutions generated along the path.

Similarly as the crossover operation, the bound set is used to check that the solution produced I_i falls in the admissible area. If so, the solution is compared with the current list of potential solutions, and a local search is performed. Otherwise, no improvement strategy is triggered, and the solution is simply ignored. Figure 7 shows a sample of the solutions produced by this operator. The main steps of this operator are summarized in Algorithm 8.

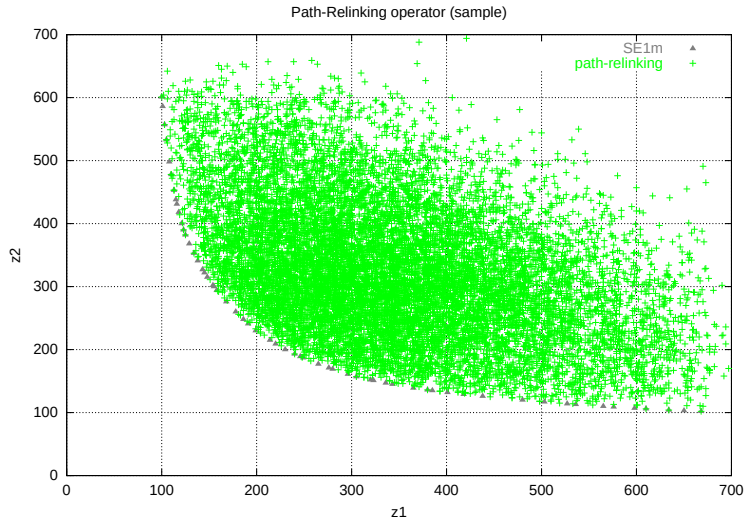

Figure 7. Sample of the solutions produced by the path-relinking operator for the `2AP100-1A20.dat` instance

2.8. The stopping rules

Three stopping rules are available in the predicate `isTheEnd?`. Each of them can be activated separately or in combination with one of the other two. The rules are based on three aspects:

- *number of iterations*. The heuristic is stopped after a predetermined effort (parameter `iterationMax`). The suggested value is 250 000 iterations;
- *a timeout*. The heuristic is stopped after a predefined elapsed time (parameter `elapsedTime`);
- *unfruitful iterations*. After a cycle of a predetermined number of iterations (parameter `seekChangesFrequency`), the rule checks to see if new elite solutions were added during the cycle (routine `noteNewSolutions`). Consecutive cycles without change are counted, and the heuristic is stopped when a predetermined number of cycles is recorded. (parameter `noChangeMax`.) The suggested value is 2 cycles of 100 000 iterations.

The most realistic stopping condition is a configuration combining one absolute value (number of iterations or timeout) with the detection of unfruitful iterations.

3. Numerical experiments

3.1. Experimental environment

A library of numerical instances for MOCO problems is available online at `www.terry.uga.edu/mcdm/`. This library contains a series of fifteen (*biAP*) instances that we used for our experiments. The name of an instance provides the following characteristics: the number of objectives, the problem, the size, the series, the objective type, and the range of coefficients in $[1, value]$. For example, `2AP5-1A20`' is a biobjective assignment problem with 5×5 variables from the series 1; the coefficients of the objective function are generated randomly (denoted by `A`) in the range $[1, 20]$.

In all of the following, the instances used are biobjective, with objective coefficients generated randomly in the range $[1, 20]$: `2AP*-1A20`'. The Instances `2AP5-1A20` through `2AP50-1A20` were also used in Tuyttens et al. (2000).

The computer used for the experiments was a laptop equipped with a PowerPC G4 400 Mhz processor, 256 Mb of RAM, running under

the OS X operating system, version 10.2.6. The algorithms were implemented in the language C. The binary code was obtained using the gcc-3.1 compiler with the -O3 optimizer option.

The default values of the parameters were fixed to implement a generation performing 250 000 iterations (parameter `iterationMax`), with the genetic map refreshed every 100 000 iterations (parameter `refreshMapFrequency`). Each experiment was repeated 5 times using different seed values. The min/avg/max values were recorded for each observed indicator. This experimental environment is common to all numerical experiments, unless explicitly stated otherwise.

The performance measure M_1 is used to appreciate and compare the approximation quality of PE. This indicator was introduced by Ulungu (1993) to measure the ratio of exact efficient solutions contained in the potential efficient solution set PE, i.e., $M_1 = |PE \bigcap E_m|/|E_m|$.

3.2. Analyses

3.2.1. *Minimal complete solution sets and initial elite solution set*

The minimal complete set of efficient solutions E_m has already been computed using Cplex on a mainframe (Degoutin and Gandibleux, 2002). This set is decomposed into the characteristic subsets, and particularly $SE1_m$ and $SE2_m$. The initial elite population set PE_{init} is computed by solving an optimal assignment with the Successive Shortest Path (SSP) algorithm, for some convex combinations of the objectives. Table II reports all the solutions computed. The CPUt column of this table provides the time consumed by the `detectPEinit` routine to compute PE_{init} on the computer used for our experiments.

According to figure 8, the number of solutions in each subset grows linearly with the input size for these instances. It is interesting to note that the difference between the sizes of $SE1_m$ and $SE2_m$ seems to decrease with input size.

The examination of the PE_{init} column confirms that PE_{init} is composed of all solutions belonging to $SE1_m$, plus some additional solutions from $SE2_m$. It is not surprising to observe an increasing difference between $SE1_m$ and the PE_{init}, in instances involving large input size. Because $SE2_m$ grows with input size, the frequence of detected solutions in $SE2_m$ using the dichotomic principle of the routine `detectPEinit` increases. The computation time of PE_{init} for instances up to 50×50 variables is small, and remains reasonable even for larger-sized instances.

Table II. Distribution of efficient solutions in E_m, and its subsets, as well as in PE_{init}.

instance					Cplex	SSP	
n×n	E_m	NE_m	SE_m	$SE1_m$	$SE2_m$	PE_{init}	CPUt
5	8	5	3	3	0	3	0.0
10	16	10	6	6	0	6	0.0
15	39	27	12	12	0	12	0.0
20	55	42	13	13	0	13	0.0
25	74	49	25	20	5	21	0.0
30	88	61	27	24	3	24	1.0
35	81	54	27	25	2	25	0.0
40	127	73	54	34	20	38	2.0
45	114	71	43	32	11	33	2.0
50	163	96	67	39	28	43	3.0
60	128	84	44	39	5	41	6.0
70	174	114	60	42	18	46	11.0
80	195	126	69	47	22	50	18.0
90	191	108	83	51	32	60	30.0
100	223	122	101	50	51	59	41.0

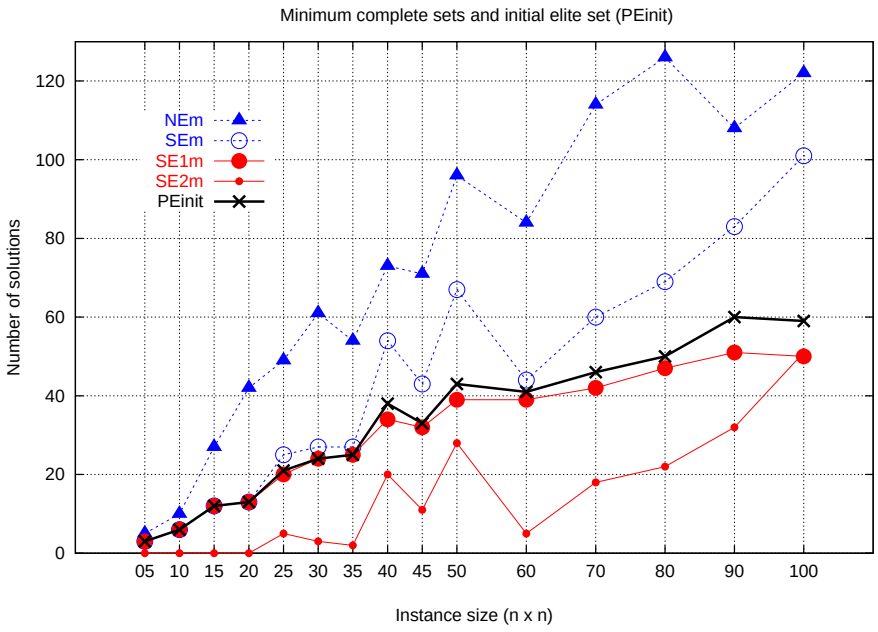

Figure 8. Minimal complete sets and the initial elite set (PE_{init})

3.2.2. *Approximations observed according to diverse stopping conditions*

Tables III, IV and V report the performances of the heuristic given the respective activation of rules 1, 2 and 3 (i.e. number of iterations, time-out and an unfruitful iterations) as stopping conditions. We used 250 000 iterations as the `iterationMax` parameter for rule 1, 60 seconds as `elapsedTime` parameter for rule 2, and two cycles of 100 000 iterations as the `noChangeMax` parameter for rule 3. In each case, two situations were evaluated depending on whether the initial elite set was initialised with SE_m, the minimal complete set of supported solutions, or with the PE_{init} detected by the routine `detectPE1init`.

Initializing the set of potential efficient solutions with a less rich population of individuals (in this case, a smaller number of solutions) does not seem to deteriorate the heuristic's ability to approximate the efficient frontier. The detection ratio curve is delayed when the population is less rich, but it preserves the same tendency. For example, when the rule 1 is activated (figure 9), the detection ratio M_1 for PE_{init} is generally worse compared with SE_m, and the difference increases with the input size of the instances. Nevertheless, the detection quality remains very good, with a difference of less than 10%. Consequently, if the time consumed for detecting an initial population set is judged inacceptable or if it becomes excessive with larger input sizes, the detection procedure can be stopped at a premature stage. Although the initial elite set will not be completed, the heuristic is still able to compute good approximations of efficient solutions with initially inferior genetic information.

For the instances up to 50×50 variables, the detection quality is practically identical no matter what stopping condition is used (see figure 10). In larger instances, the differences are visible. Three observations can be made. First, the detection curves for all three rules exhibit the same behavior. Second, even given a very short computing time, our heuristic is able to find more than 50% of exact solutions for the largest instance. Third, allowing the heuristic to reiterate until there are no new candidate solutions for the set PE produces a final approximation containing a larger number of exact efficient solutions. This last observation indicates that the algorithm tends to progress towards the exact solution set, although it may require a significant amount of CPUt time to reach it.

3.2.3. *The impact of heuristic components on the generation process*

The proposed algorithmic solution (see algorithm 1) is built on four main components: crossover, path-relinking, local search on new elites and local search in a neighborhood. The following analysis shows how

Table III. M_1(min/avg/max) and CPUt (avg) using SE_m and PE_{init} for rule 1

instance	SE_m				PE_{init}			
n×n	min	avg	max	CPUt	min	avg	max	CPUt
5	100.0	100.0	100.0	1.6	100.0	100.0	100.0	2.0
10	100.0	100.0	100.0	4.2	100.0	100.0	100.0	4.2
15	100.0	100.0	100.0	13.8	100.0	100.0	100.0	13.8
20	100.0	100.0	100.0	25.4	100.0	100.0	100.0	25.4
25	90.5	94.6	97.3	21.0	90.5	93.2	95.9	21.0
30	93.3	94.8	96.6	35.6	94.4	96.4	97.8	37.4
35	95.1	96.0	96.3	41.0	95.1	95.3	96.3	40.6
40	94.5	96.6	98.4	43.0	87.5	89.2	92.2	49.2
45	85.1	87.7	91.2	47.8	83.3	87.2	91.2	57.4
50	89.7	90.3	91.5	69.4	85.5	87.0	88.5	82.4
60	76.6	78.4	80.5	85.4	69.5	73.3	77.3	88.2
70	77.0	80.5	82.8	121.4	70.7	71.7	74.1	128.6
80	74.9	77.1	79.5	165.2	74.9	77.8	81.0	175.4
90	72.3	75.3	77.0	198.0	71.2	71.9	72.3	200.4
100	71.3	72.4	74.4	254.0	60.1	62.5	63.7	292.4

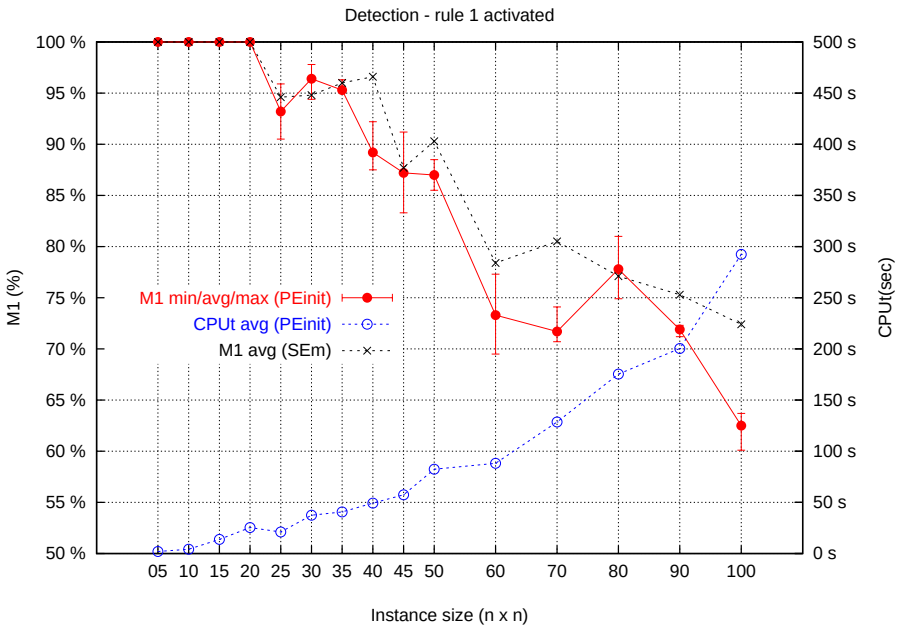

Figure 9. M_1(min/avg/max) and CPUt (avg) on SE_m and PE_{init} for rule 1

Table IV. M_1(min/avg/max) and #gen (avg) using SE_m and PE_{init} for rule 2

instance	SE_m				PE_{init}			
n×n	min	avg	max	#gen	min	avg	max	#gen
5	100.0	100.0	100.0	5295096	100.0	100.0	100.0	5080107
10	100.0	100.0	100.0	2951570	100.0	100.0	100.0	2891947
15	100.0	100.0	100.0	1051004	100.0	100.0	100.0	1022942
20	100.0	100.0	100.0	581343	100.0	100.0	100.0	568462
25	95.9	97.6	98.6	709546	95.9	96.8	98.6	676334
30	95.5	96.0	96.6	419831	95.5	97.1	98.9	388096
35	96.3	96.3	96.3	368043	95.1	95.8	96.3	359005
40	94.5	96.7	98.4	349506	87.5	90.2	93.8	296375
45	87.7	88.8	91.2	314674	83.3	87.2	91.2	252853
50	89.7	90.2	91.5	218105	84.2	85.3	86.7	179193
60	71.9	74.1	75.8	180284	66.4	71.7	75.0	167976
70	71.8	73.9	75.3	124785	64.9	67.5	70.1	116007
80	67.7	71.6	73.8	92912	70.3	71.4	72.3	85111
90	68.1	69.2	70.7	77062	59.7	62.8	64.9	73591
100	61.4	64.1	66.8	59407	48.4	49.7	50.2	50648

Table V. M_1(min/avg/max) and CPUt (avg) using SE_m and PE_{init} for rule 3

instance	SE_m				PE_{init}			
n×n	min	avg	max	CPUt	min	avg	max	CPUt
5	100.0	100.0	100.0	1.0	100.0	100.0	100.0	1.0
10	100.0	100.0	100.0	3.6	100.0	100.0	100.0	2.6
15	100.0	100.0	100.0	11.8	97.4	99.5	100.0	10.0
20	100.0	100.0	100.0	19.6	96.4	98.9	100.0	21.8
25	90.5	95.4	98.6	28.0	89.2	92.4	95.9	20.0
30	93.3	94.6	96.6	36.4	94.4	96.2	97.8	33.2
35	95.1	96.0	96.3	48.6	95.1	95.1	95.1	36.4
40	94.5	96.6	98.4	56.8	88.3	92.7	94.5	107.2
45	86.0	88.4	91.2	67.4	84.2	87.9	93.0	85.2
50	90.3	91.9	92.7	146.8	87.3	88.8	89.7	179.2
60	82.0	84.5	86.7	214.4	79.7	81.9	85.2	266.8
70	81.0	86.1	90.8	325.8	71.8	77.4	82.8	457.0
80	79.0	81.3	84.1	455.0	80.5	83.5	86.7	541.2
90	77.0	80.8	84.3	666.4	77.0	79.6	83.2	777.6
100	78.0	79.6	81.6	1030.8	65.5	69.3	74.0	1166.6

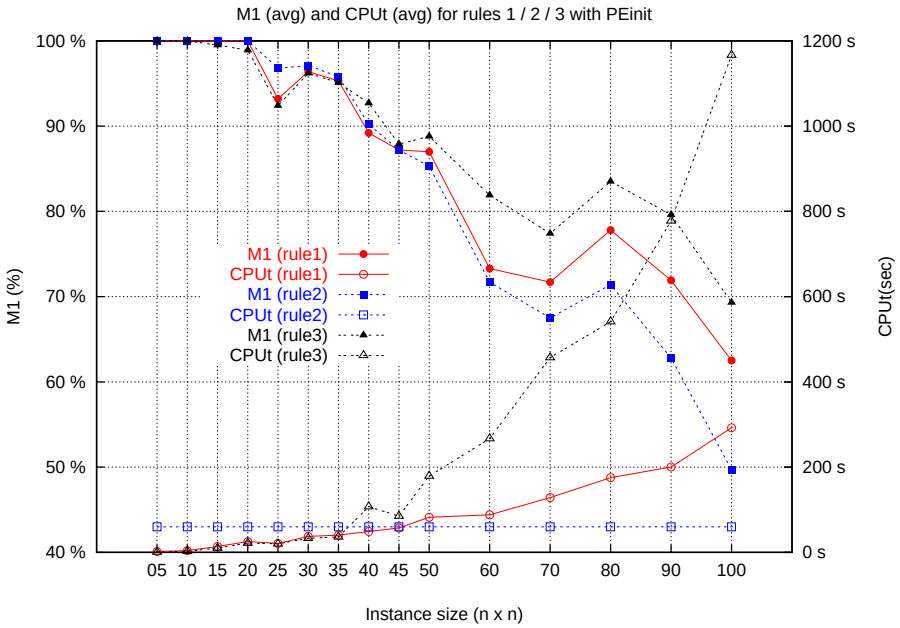

Figure 10. M_1(avg) and CPUt (avg) on SE_m and PE_{init} for rules 1, 2 and 3

each component participates in the progress of detecting exact efficient solutions. To achieve this analysis, four configurations of the heuristic were tested using `2AP100-1A20`. In each configuration, one component of the heuristic was disabled. Notice that the four components are not independent, although crossover and path-relinking are independent of each other. For example, all crossover, path-relinking and local search in a neighborhood call the procedure of local search on new elites (see Algorithms 5, 6 and 8). Table VI reports values for M_1 and CPUt at the end of the heuristic (rule 1 activated; 250 000 iterations). Figure 11 shows the progression curves for the intermediate steps.

The crossover operator contributes particularly to the detection of good solutions during the early iterations of the generation process. However, crossover seems less powerful than path-relinking. Indeed, the time spent by the heuristic when crossover or path-relinking is disabled is of the same order (152s and 186s respectively). However, the heuristic is definitely less effective when path-relinking is disabled (a final M_1 value of 42% compared to 62%). Analysis shows that the crossover operation contributes to the detection of new potential efficient solutions, but its impact on detection fades as iterations increase. One explanation is that the genetic map loses its information capacity as the PE set becomes larger.

The local search on new elites pushes the good solutions towards the efficient frontier. This operator is especially effective during early iterations of the generation, because many new solutions are detected. However, it becomes much less effective in the latter half of iterations.

The pair, 'path-relinking' and 'local search in a neighborhood', is clearly a powerful instigator of solutions within the heuristic. When either of these two components is disabled, the corresponding decrease in detection power of the heuristic is similar in proportion. Together, these two components seem to be sufficient to constitute a simplified version of the heuristic. Nevertheless the 'crossover' and 'local search on new elites' components appear to be good complements for the heuristic.

Table VI. Progression on M_1 for instance `2AP100-1A20` with 250 000 iterations

component disabled in the heuristic	M_1 (%)	CPUt (s)
none	61.88	336.00
crossover operator	61.88	152.00
path-relinking operator	42.15	186.00
local search on new elites operator	62.33	335.00
local search in a neighborhood	44.84	252.00

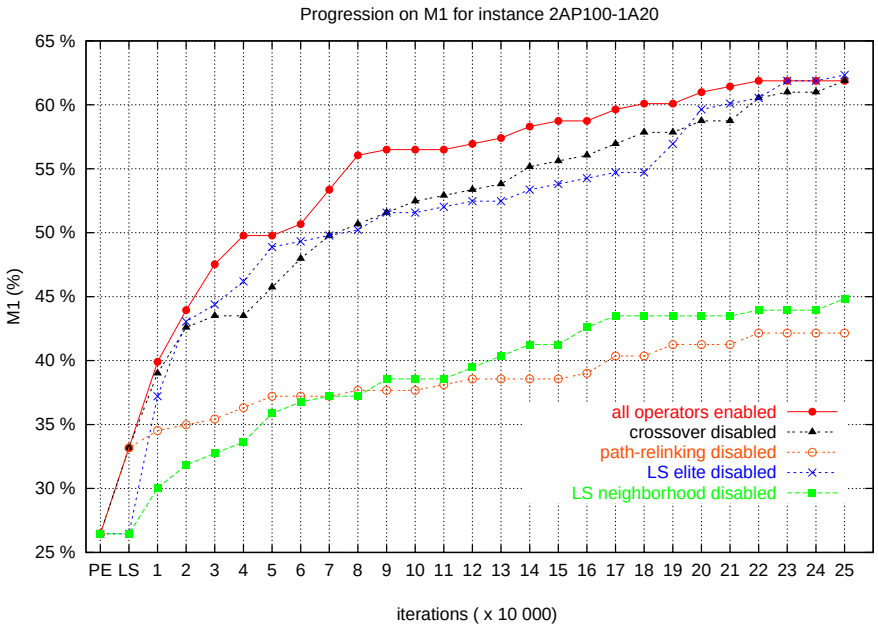

Figure 11. Detection ratios for the cases with different sets of operators enabled

3.2.4. *Our final results compared with those existing in the litterature*
The reported computational results in column two of table VII are taken from the results published in Tuyttens et al. (2000). They are obtained for one run only using the improved MOSA method with the same set of numerical instances. Figure 12 shows that MOSA method performs poorly in terms of the quality measure M1, especially when the size of the instances increases. Since the computer used is not the same (MOSA results have been computed on DEC3000 alpha), a discussion in term of CPUt is not allowed.

The columns on the right-hand side of table VII report the results of our heuristic when rules 1 and 3 are activated. The generation is tuned to 250 000 iterations and the genetic map is refreshed each 100 000 iterations. The CPUt indicated is an average value for one complete run of the heuristic. It includes the time used for computing the initial population of elite solutions, and the time used for the approximation of the efficient frontier. The rightmost column of the table gives the average number of iterations performed by the algorithm during a generation. A value other than 250 000 indicates that rule 3 was triggered before rule 1.

The comparison of MOSA results with those produced by the population based heuristic proposed in this paper underlines the superiority of our heuristic over MOSA (figure 12). We suppose that our heuristic consumes more time than the MOSA method (CPUt for MOSA are reported as 5s and 246s respectively for instances 5×5 and 50×50). However, our heuristic has two main characteristics that persuade us that in tests run on the same computer, our method would outperform the MOSA method. First, the detection of solutions evolves very quickly during the early iterations of the generation. Despite the short time allowed for the generation, the approximation quality is already good (see table IV). Second, our heuristic is able to improve its approximation if more time is allowed (see table V). MOSA on the other hand, does not seem to be able to improve its approximation even given more time. According to the values reported for the quality indicator M_2 in Tuyttens et al. (2000), the MOSA method rapidly encounters difficulties for detecting good approximations of the efficient frontier.

4. Conclusion

We have described a population-based heuristic for solving assignment problems with two objectives. This heuristic uses three operators - crossover, path-relinking and local search - on a population composed only of elite solutions. Based on simple principles, our heuristic is easy

Table VII. MOSA vs the population-based heuristic with rules 1 and 3

instance	$MOSA$					PE_{init}	
n×n	M_1			M_1		CPUt	#iter
		min	avg	max		avg	avg
5	87.5	100.0	100.0	100.0		0.6	100000
10	56.2	100.0	100.0	100.0		2.6	150000
15	25.6	97.4	99.5	100.0		10.2	180000
20	3.7	96.4	98.9	100.0		19.8	190000
25	0.0	89.2	92.4	95.9		18.6	210000
30	3.4	94.4	96.2	97.8		33.6	210000
35	0.0	95.1	95.1	95.1		37.4	220000
40	0.0	87.5	89.2	92.2		53.6	250000
45	0.0	83.3	86.3	91.2		59.6	240000
50	0.0	85.5	87.0	88.5		89.0	250000
60	N.A.	69.5	73.3	77.3		98.8	250000
70	N.A.	70.7	71.7	74.1		146.0	250000
80	N.A.	74.9	77.8	81.0		202.0	250000
90	N.A.	71.2	71.9	72.3		240.8	250000
100	N.A.	60.1	62.5	63.7		347.4	250000

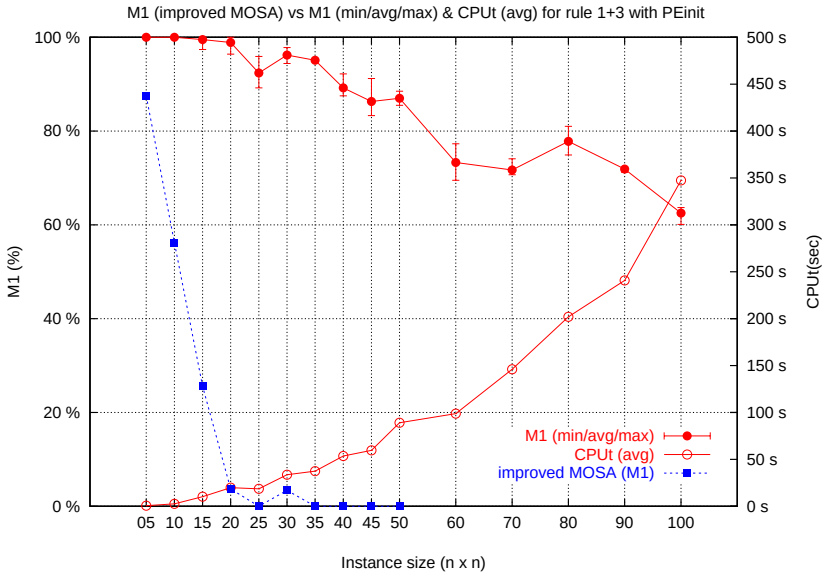

Figure 12. The population based heuristic with rules 1-3 activated, compared with MOSA

to implement and involves only two parameters that do not require any tuning phase.

In a first step, a set of efficient supported solutions is computed. This set composes the initial population of elite solutions. In a second step, this population undergoes a generation process involving the three operators. The heuristic is able to detect a good approximation of the efficient frontier even given a short computing time (i.e. a small number of iterations). This is confirmed by experiments that show good performances on numerical instances, especially as compared to the results previously obtained with other heuristics.

Several perspectives for further research are possible. First, in order to handle larger instances more efficiently, it may be useful to divide the genetic information into sectors along the efficient frontier, according to a principle of clusters. Such a technique could help to maintain a representative genetic map locally and avoid genetic map sterilized by too many diverse solutions. Another promising track consists in designing of a less random path-relinking operator. Rather than building only one neighbor at random for each stage of the path, the generation of multiple neighbors according to a given characteristic, and the selection of a good neighbor from among them, could make path-relinking even more powerful. A third possibility concerns the local search. In the current version, a local search is systematically applied to a solution when it is included in the promising zone. A same solution can be visited several times and thus to generate several times the same neighbors. An improved filtering of the solutions that must undergo a local search would contribute to reduce the computation time. Lastly, the resolution ability of our heuristic could be confirmed by further experimentations with numerical instances exhibiting contrasted characteristics.

References

R. K. Ahuja, T. L. Magnanti, and J. B. Orlin, *Network flows: Theory, Algorithms, and Applications*, Prentice-Hall, 1993.

C. Coello, D. Van Veldhuizen and G. Lamont. *Evolutionary Algorithms for Solving Multi-Objective Problems*. Kluwer Academic Publishers, New York, 2002.

F. Degoutin and X. Gandibleux, *Un retour d'expériences sur la résolution de problèmes combinatoires bi-objectifs*, 5e journée du groupe de travail Programmation Mathématique MultiObjectif (PM20), Angers, France, mai 2002.

M. Dorigo and G. Di Caro, The Ant Colony Optimization Meta-Heuristic. In D. Corne, M. Dorigo and F. Glover, editors, *New Ideas in Optimization*, McGraw-Hill, 11-32, 1999.

M. Ehrgott and X. Gandibleux, Bounds and Bound Sets for Biobjective Combinatorial Optimization Problems. In *Multiple Criteria Decision Making in the New*

Millennium (M. Koksalan and St. Zionts Eds.). Lecture Notes in Economics and Mathematical Systems 507, pp.241-253, Springer, 2001.

M. Ehrgott and X. Gandibleux, Multiobjective Combinatorial Optimization. In *Multiple Criteria Optimization: State of the Art Annotated Bibliographic Survey* (M. Ehrgott and X. Gandibleux Eds.), pp.369-444, Kluwer's International Series in Operations Research and Management Science : Volume 52, Kluwer Academic Publishers, Boston, 2002.

X. Gandibleux, H. Morita, N. Katoh, The Supported Solutions used as a Genetic Information in a Population Heuristic. In *Evolutionary Multi-Criterion Optimization* (E. Zitzler, K. Deb, L. Thiele, C. Coello, D. Corne Eds.). Lecture Notes in Computer Sciences 1993, pp.429-442, Springer, 2001.

X. Gandibleux, H. Morita, N. Katoh, Use of a genetic heritage for solving the assignment problem with two objectives. In *Evolutionary Multi-Criterion Optimization* (C. Fonseca, P. Fleming, E. Zitzler, K. Deb, L. Thiele Eds.). Lecture Notes in Computer Sciences 2632, pp 43-57, Springer, 2003a.

X. Gandibleux, H. Morita, N. Katoh, Impact of clusters, path-relinking and mutation operators on the heuristic using a genetic heritage for solving assignment problems with two objectives. *MIC2003 Fifth Metaheuristics International Conference*, Kyoto, Japan, August 25 - 28, 2003b.

X. Gandibleux, M. Sevaux, K. Sörensen and V. T'kindt (Eds.) *Multiople Objective Metaheuristics* Proceedings of the workshop "MOMH: Multiople Objective MetaHeuristics", November 04-05, 2002, Carré des Sciences, Paris. Lecture Notes in Economics and Mathematical Systems, Springer (in press).

F. Glover and M. Laguna. *Tabu search*. Kluwer Academic Publishers, Boston, 1997.

P. Hansen. Bicriterion path problems. In *Multiple Criteria Decision Making Theory and Application* (G. Fandel and T. Gal Eds.). Lecture Notes in Economics and Mathematical Systems 177, pp 109–127. Springer Verlag, Berlin, 1979.

R. Malhotra, H.L. Bhatia and M.C. Puri, Bi-criteria assignment problem, *Operations Research*, 19(2): 84-96,1982.

H. Morita, X. Gandibleux, N. Katoh, Experimental feedback on biobjective permutation scheduling problems solved with a population heuristic. *Foundations of Computing and Decision Sciences Journal*, pp 23-50, Vol.26, No1, 2001.

D. Tuyttens, J. Teghem, Ph. Fortemps and K. Van Nieuwenhuyse, Performance of the MOSA method for the bicriteria assignment problem, *Journal of Heuristics*, 6 pp. 295–310, (2000).

E.L. Ulungu, *Optimisation Combinatoire multicritère: Détermination de l'ensemble des solutions efficaces et méthodes interactives*, Université de Mons-Hainaut, Faculté des Sciences, 313 pages, 1993.

Algorithm 1 The main procedure

Require: data : c_{il}^q ; $i = 1, n$; $j = 1, n$; $q = 1, 2$; iterationMax
Ensure: PE

```
--| Compute the initial elite population set PE_init
detectPEinit( data ↓ , peInit ↑) ; pe ← peInit
--| Compute the nadir points
buildUpperBoundSet( pe ↓ , nadirs ↑ )
--| A first local search on the SE1_m solution set
localSearchOverNewElites( pe ↕ )
--| Identify the genetic heritage and elaborate the genetic map
elaborateGeneticInformation( pe ↓ , map ↑)

--| Initialize the running indicators
iteration ← 1 ; elapsedTime ← 0 ; changes ← 0 ; noMore ← 0
repeat

  --| Elaborate a solution by crossover
  crossoverWithElites( pe ↕ , map ↓ , nadirs ↓ )

  --| Elaborate a series of solutions by path-relinking
  pathRelinkingWithElites( pe ↕ , nadirs ↓ )

  --| Apply a local search to the new elite solutions in PE
  localSearchOverNewElites( pe ↕ )

  --| Refresh the genetic heritage by integrating genetic information
  --| coming from the new PE in the current map
  if (iteration MOD refreshMapFrequency = 0) then
    elaborateGeneticInformation( pe ↓ , map ↕ )
  end if

  --| Identify the producer of new potential solutions and note
  --| a series of iterations without production of new PE
  noteNewSolutions( pe ↓ , changes ↑ )
  if (iteration MOD seekChangesFrequency = 0) then
    noMore ← (changes = 0 ? noMore + 1 : 0) ; changes ← 0
  end if

  --| Check the stopping condition(s)
until isTheEnd?( iteration++ ↓ , elapsedTime ↓ , noMore ↓ )
```

Algorithm 2 Procedure `detectPEinit`

--| Compute $x^{(1)}$ and $x^{(2)}$, the optimal solutions for the two objectives
x1 ← `succShortestPath`($c^1 \downarrow$, data $\downarrow$)
x2 ← `succShortestPath`($c^2 \downarrow$, data $\downarrow$)
L ← { x1 , x2 }

--| Compute all intermediate solutions between $x^{(1)}$ and $x^{(2)}$
`solveRecursion`(x1 $\downarrow$, x2 $\downarrow$, data $\downarrow$, L $\updownarrow$)

--| Remark : $x^{(1)}$ and $x^{(2)}$ can be "weakly" non-dominated.
peInit ← `removeDsolutions`(L)

Algorithm 3 Procedure `solveRecursion`

--| Compute the optimal solutions $x^{(C)}$ of (P_λ) :
--| $\min\{\lambda_1 z^1(x) + \lambda_2 z^2(x) \mid x \in X\}$
--| where $\lambda_1 = z^2(x^{(A)}) - z^2(x^{(B)})$, and $\lambda_2 = z^1(x^{(B)}) - z^1(x^{(A)})$.
$c^\lambda_{ij} = \lambda_1\ c^1_{ij} + \lambda_2\ c^2_{ij}$, i=1,n j=1,n
xC ← `succShortestPath`($c^\lambda \downarrow$, data $\downarrow$)

if (z(xC) ≠ z(xA)) **and** (z(xC) ≠ z(xB)) **then**
  L ← L ∪ { xC }
  `solveRecursion`(xA $\downarrow$, xC $\downarrow$, data $\downarrow$, L $\updownarrow$)
  `solveRecursion`(xC $\downarrow$, xB $\downarrow$, data $\downarrow$, L $\updownarrow$)
end if

Algorithm 4 Procedure `SeekInNeighborhood`

for j1 **in** 1..n-1 **do**
  for j2 **in** j1 + 1..n **do**
    I' ← swap(I_{j1} , I_{j2}) --| giving a new neighbor
    if `isNonDominated`(I' , pe) **and** (z(I') ≠ z(I), ∀ I ∈ pe) **then**
      --| I' is a new unique elite solution
      pe ← Add/update pe with I'
    end if
  end for
end for

Algorithm 5 Procedure `localSearchOverNewElites`

```
--| Perform a local search on all the last added solutions in pe
loop
  lastAdded ← set of all the last added solutions in pe
  exit when lastAdded = ∅
  for all I ∈ lastAdded do
    seekInNeighborhood( I , pe )
  end for
end loop
```

Algorithm 6 Procedure `crossoverWithElites`

```
--| Selection of individuals
Select I1 ∈ pe and I2 ∈ pe at random, I1 ≠ I2

--| An offspring I3 is built from individuals I1 and I2 by using
--| the genetic information coming from I1, I2 and the PE set
for j in 1..n do
  --| Common genes are copied
  if  I1_j = I2_j  then
    I3_j ← I1_j
  end if
  --| Others genes : value is selected using the wheel
  if I1_j ≠ I2_j then
    I3_j ← useGeneticInformation( j ↓ , map ↓ )
  end if
end for

--| Check if I3 is a new promising sol.; if so, to perform a local search
--| if I3 is outside of the promising area, then it is ignored
if isNonDominated( I3 , nadirs ) then
  --| I3 is located in the admissible area for performing a LS
  if isNonDominated( I3 , pe ) and ( z(I3) ≠ z(I), ∀ I ∈ pe ) then
    --|  I3 is a new unique elite solution
    pe ← Add/update pe with I3
  end if
  SeekInNeighborhood( I3 , pe )
end if
```

Algorithm 7 Function `useGeneticInformation`

```
--| Assignment i for position j is selected using the map
--| composed of the roulette wheels
i ← selectAssignmentUsingRouletteWheel( j , map )

if the assignment ( i , j ) is not feasible then
  LFT ← elaborate a list of feasible tasks for position j
  i ← selectFeasibleAssignmentRandomly( LFT )
end if
return i
```

Algorithm 8 Procedure `pathRelinkingWithElites`

```
--| Selection of individuals
Select I1 ∈ pe and I2 ∈ pe at random, I1 ≠ I2

--| Elaborate a path between I1(initiating solution) and
--| I2 (guiding solution)
while I1 ≠ I2 do
  --| Perform a move from I1 to I2
  LDA ← list of difference in assignments between I1 and I2
  i ← random(1...card(LDA))
  taskI2 ← taskOf*InPosition*( I2 , i )
  j ← positionOf*InIndividual* ( taskI2 , I1 )
  I1_j ← taskOf*InPosition*( I1 , i )
  I1_i ← taskI2

  --| Check if I1 updated is a new promising sol.; if so, to perform a
  --| local search. If I1 is not in the promising area, it is ignored
  if isNonDominated( I1 , nadirs ) then
    --| I1 is located in the admissible area for performing a LS
    if isNonDominated( I1 , pe ) and ( z(I1) ≠ z(I), ∀ I ∈ pe ) then
      --|  I1 is a new unique elite solution
      pe ← Add/update pe with I1
    end if
    SeekInNeighborhood( I1 , pe )
  end if

end while
```