On local search for bi-objective knapsack problems

Arnaud Liefooghe, Luis Paquete, José Rui Figueira

On Local Search for Bi-objective Knapsack Problems

Arnaud Liefooghe arnaud.liefooghe@univ-lille1.fr
LIFL, Université Lille 1, UMR CNRS 8022, 59655 Villeneuve d'Ascq cedex, France

Luís Paquete paquete@dei.uc.pt
CISUC, Department of Informatics Engineering, Faculty of Science and Technology, University of Coimbra, 3030-290 Coimbra, Portugal

José Rui Figueira figueira@ist.utl.pt
CEG-IST, Instituto Superior Técnico, 1040-001 Lisboa, Portugal
(Associate Researcher at LORIA Laboratory, Nancy, France)

Abstract

In this article, a local search approach is proposed for three variants of the bi-objective binary knapsack problem, with the aim of maximizing the total profit and minimizing the total weight. First, an experimental study on a given structural property of connectedness of the efficient set is conducted. Based on this property, a local search algorithm is proposed and its performance is compared against exact algorithms in terms of running time and quality metrics. The experimental results indicate that this simple local search algorithm is able to find a representative set of optimal solutions in most of the cases, and in much less time than exact algorithms.

Keywords

Multi-objective combinatorial optimization, population-based local search, structural analysis, connectedness, knapsack problems.

1 Introduction

Problems arising in multi-objective combinatorial optimization (MoCO) are difficult to solve. For most MoCO problems, the number of efficient solutions is very large, and determining if a feasible solution is optimal is NP-complete (Ehrgott, 2000). Learning about the problem structure helps to understand those difficulties and to design better algorithms. Stochastic local search algorithms have been successfully applied to many MoCO problems. It is widely accepted that their good performance is related to some structural properties of the search space that allow local search procedures to find reasonably good quality solutions in an effective manner. However, little is known about which properties these are, and how they can affect the performance of the class of multi-objective local search algorithms.

In this article, the notion of *connectedness* (Ehrgott and Klamroth, 1997) of the set of efficient solutions for MoCO problems (also known as *the efficient set*) is analyzed from an experimental point of view. The main results of this analysis is then related to the performance of a particular class of multi-objective local search algorithms. Specifically, a Pareto-based approach, combining local search principles with the use of a population of solutions, is investigated. It is based on the Pareto Local Search (PLS) algorithm (Paquete et al., 2007). PLS is a simple and straightforward extension of local search for

the multi-objective case. Existing algorithms from evolutionary multi-objective optimization share similar principles with PLS, like PAES (Knowles and Corne, 2000) or SEMO (Laumanns et al., 2004). Those algorithms combine the definition of a neighborhood structure with the management of an archive of potentially efficient solutions found so far. This archive is iteratively improved by exploring the neighborhood of its own content until no further improvement is possible, or another stopping condition is satisfied. A detailed explanation of the principles of these algorithms can be found in (Liefooghe et al., 2011a).

For a given efficient set of a MoCO problem instance, a graph can be constructed such that each node represents an efficient solution and an edge connects two nodes if the corresponding solutions are neighbors for a given neighborhood structure. The efficient set is connected with respect to that neighborhood structure if the underlying graph is also connected, that is, there is a path between any pair of nodes. If the efficient set is connected and the neighborhood structure is tractable from a computational point of view, local search algorithms would be able to find the efficient set in a very effective manner by starting with at least one efficient solution. Moreover, such a local search approach has a strong advantage since it provides solutions both in objective and decision space. However worst-case results have shown that the efficient set for many MoCO problems is not connected in general with respect to different neighborhood structures (Ehrgott and Klamroth, 1997; Gorski et al., 2011), except for very few particular cases, including bi-objective knapsack problems with constant sum and equal weighted items (Gomes da Silva et al., 2004), three-objective knapsack problems with binary weights (Gorski et al., 2012), as well as bi-objective matroid problems with at least one binary sum objective (Gorski, 2010). Moreover, some recent results indicate that approximate solutions that are obtained from independent metaheuristic runs on the bi-objective traveling salesman problem are strongly clustered with respect to small-sized neighborhood structures (Paquete and Stützle, 2009), *i.e.* there exists very few connected components, and most efficient solutions belong to the same component. In terms of problem-solving, the present results may be relevant for other applications and other Pareto local search variants or hybrid algorithms (Dubois-Lacoste et al., 2011; Liefooghe et al., 2011a; Lust and Teghem, 2010).

Recently, we analyzed the connectedness of two bi-objective knapsack problem formulations (Liefooghe et al., 2011b). In this article, we extend the analysis to an additional variant, we examine all problems in detail and we give more insights on the experimental study. In particular, the connectedness property is here investigated experimentally for three variants of the bi-objective binary knapsack problem: The bi-objective unconstrained knapsack problem (BUKP), the bi-objective knapsack problem with bounded cardinality (BKP-BC), and the bi-objective knapsack problem with fixed cardinality (BKP-FC). Those problems are all NP-hard and intractable in the general case (Ehrgott, 2000). The experimental results suggest that the efficient set for the three problems is very often connected with respect to elementary neighborhood structures, despite of the negative results for the general problem (Gorski et al., 2011). Based on these positive findings, we adapt a local search algorithm that exploits this property and we compare its performance with dynamic programming (DP) algorithms in terms of running-time, number and quality of solutions found. A special technique is introduced that allows the early termination of the complete neighborhood exploration without harming algorithmic performance in terms of solution quality.

The article is organized as follows. The three variants of the bi-objective knapsack problem (*i.e.* BUKP, BKP-BC and BKP-FC) are presented in Section 2. Section 3

provides a connectedness analysis for BUKP, BKP-BC and BKP-FC, respectively. Moreover, the section presents the exact approaches for the corresponding problems too. The local search algorithm is introduced in Section 4, together with numerical results obtained on a large set of bi-objective knapsack problem instances of different structure and size. Finally, Section 5 presents conclusions and further work.

2 Bi-objective Knapsack Problems

This section introduces three variants of the binary bi-objective knapsack problem. Some definitions related to MoCO are given, and we detail the set of problem instances that will be used in the experiments.

2.1 Knapsack Problems

The original (single-objective) $0/1$ knapsack problem can be stated as follows.

$$\begin{aligned} \max \quad & \sum_{i=1}^{n} p_i x_i \\ \text{s.t.} \quad & \sum_{i=1}^{n} w_i x_i \leqslant W \end{aligned} \tag{KP}$$

where $p = (p_1, p_2, \ldots, p_i, \ldots, p_n)$ is the profit vector, $p_i \in \mathbb{N}$ representing the amount of profit on item i, $i \in \{1, \ldots, n\}$, and $x = (x_1, x_2, \ldots, x_i, \ldots, x_n)$ with $x_i = 1$ if the item i is included in the subset of selected items (the knapsack) and $x_i = 0$ otherwise; $w = (w_1, w_2, \ldots, w_i, \ldots, w_n)$ is the weight vector, $w_i \in \mathbb{N}$ representing the amount of investment on item i, $i \in \{1, \ldots, n\}$; and W is the overall amount available or budget. The sum of profits and the sum of weights of a given solution x are denoted by $p(x)$ and $w(x)$, respectively.

2.2 The Bi-objective Unconstrained Knapsack Problem

By transforming the capacity constraint of Problem (KP) into an objective function, the following bi-objective unconstrained knapsack problem (Ehrgott, 2000), BUKP for short, can be obtained as follows.

$$\text{vmax} \quad \{p(x), -w(x)\} \tag{BUKP}$$

A proper meaning to the operator "vmax" of the problem above is given as follows. Let X denote the set of feasible solutions of Problem (BUKP). The image of the feasible solutions when using the vector maximizing function defines the feasible region in the objective space, denoted here by $Z \subseteq \mathbb{N}^2$. A feasible solution $x \in X$ is *efficient* if there does not exist another feasible solution $x' \in X$ such that $p(x') \geq p(x)$ and $w(x') \leq w(x)$, with at least one strict inequality in one of above (or $(p(x'), -w(x')) \geq (p(x), -w(x))$). A vector $z \in Z$ is *nondominated* if there is some efficient solution $x \in X$ such that $z = (p(x), -w(x))$. A vector $z \in Z$ *dominates* a vector $z' \in Z$ (or z' *is dominated* by z) if $z \geq z'$ holds. If neither $z \not\geq z'$ nor $z' \not\geq z$ holds, then both are *mutually nondominated*. The set of all efficient solutions and the set of nondominated vectors are called the *efficient set* and the *nondominated set*, respectively. One of the most challenging issues related to MoCO relies on the identification of a *minimal complete efficient set*, that is, the smallest subset of the efficient set whose image coincides with the nondominated set. Note that this subset may not be unique, since multiple solutions can map to the same nondominated vector.

Table 1: Instances investigated in the article for the experimental analysis.

Description	Parameter	Set of values
Problem size	n	$\{100, 200, \ldots, 1000\}$
Data correlation	ρ	$\{-0.8, -0.4, 0.0, 0.4, 0.8\}$
Cardinality bound	k	$\{n/10, n/5, n/2\}$

2.3 The Bi-objective Knapsack Problem with Bounded Cardinality

The bi-objective knapsack problem with bounded cardinality (BKP-BC) extends the BUKP that is obtained by limiting the number of chosen items by a cardinality bound (k). The formulation of BKP-BC is as follows.

$$\begin{aligned} \text{vmax} \quad & \{p(x), -w(x)\} \\ \text{s.t.} \quad & \sum_{i=1}^{n} x_i \leqslant k \end{aligned} \tag{BKP-BC}$$

where k gives the maximum number of items allowed in the knapsack. In the problem above, the operator "vmax" follows the same meaning as in Problem (BUKP). The same terminology and notation will be used for this problem.

2.4 The Bi-objective Knapsack Problem with Fixed Cardinality

Finally, the bi-objective knapsack problem with fixed cardinality (BKP-FC) is a variant of BKP-BC where the cardinality is fixed, *i.e.* a fixed number of items must be in the knapsack.

$$\begin{aligned} \text{vmax} \quad & \{p(x), -w(x)\} \\ \text{s.t.} \quad & \sum_{i=1}^{n} x_i = k \end{aligned} \tag{BKP-FC}$$

The same terminology and notation is also used for this problem.

2.5 Problem Instances

BUKP instances are defined with two parameters: problem size (n) and correlation between profit and weight vectors (ρ). Both parameters influence the size of the efficient set. The corresponding results are given as supplementary material (Liefooghe et al., 2012). The positive (negative, respectively) data correlation increases (decreases, respectively) the degree of conflict between the two objectives. The size of the instances and the correlations are given in Table 1. Profit and weight integer values are randomly generated according to a uniform distribution in $[100, 1000]$. The generation of correlated data follows the procedure given by Verel et al. (2011), based on a multivariate uniform law of dimension 2 and a correlation coefficient. In addition to n and ρ, several different values for the cardinality bound (k) are considered for BKP-BC and BKP-FC. They are given in Table 1. For each problem size, data correlation and (if apply) cardinality bound, we consider 30 different and independent instances generated at random.

3 Connectedness Analysis

This section reports an experimental analysis on the connectedness property of the efficient set for the three problems investigated in the article.

3.1 Neighborhood Structures

First, let us introduce the neighborhood structures considered for the experimental analysis: the 1-*flip*, the 1-*exchange* and the 1-*flip-exchange* neighborhoods. Two feasible solutions are 1-*flip* neighbors if they differ exactly on one bit. In other words, a given neighbor can be reached by adding or removing one item from the current solution. Hence, this neighborhood structure is directly related to the Hamming distance between binary strings. As for the 1-*exchange* neighborhood, two feasible solutions are 1-*exchange* neighbors if one can be obtained from the other by exchanging two items. The 1-*flip-exchange* neighborhood is an extension of the two neighborhoods above. Two feasible solutions are 1-*flip-exchange* neighbors if one can be obtained from the other by exchanging two items, adding one item, or removing one item. The use of 1-*exchange* and 1-*flip-exchange* neighborhoods for the problems with cardinality constraint is motivated by the assumption that many efficient solutions have the same cardinality. The size of the neighborhood is linear with n for the 1-*flip* neighborhood structure, while it is quadratic in the case of 1-*exchange* and 1-*flip-exchange*. Those neighborhood structures are related to the ones used by Gorski et al. (2011) for a similar class of knapsack problems.

3.2 Connectedness Analysis for BUKP

This section describes an experimental analysis for investigating the influence of the problem size and the degree of conflict between the two objectives on the connectedness property of the efficient set for BUKP. A multi-objective dynamic programming algorithm (MDP-BUKP) is implemented to compute the efficient set. This algorithm consists of the first phase of the Nemhauser-Ullman algorithm for the single-objective binary knapsack problem (Nemhauser and Ullman, 1969). It has been shown to be theoretically efficient for several input data distributions (Beier and Vöcking, 2004). The MDP-BUKP sequential process consists of n stages. At any stage $i \in \{1, \ldots, n\}$, the algorithm generates a set of states S_i, which represents *promising* feasible solutions made up of the first i items, $i \in \{1, \ldots, n\}$. A state $s = (s^p, s^w) \in S_i$ represents a feasible solution of profit s^p and weight s^w. The MDP-BUKP algorithm follows the recursion:

$$S_i := \text{vmax}\,\{(s^p + p_i, s^w - w_i), s \in S_{i-1}\}$$

for $i \in \{1, \ldots, n\}$, with the basis case $S_0 := \{(0, 0)\}$. Operator "vmax" returns the states that are mutually nondominated in S_i. At the last stage n, the set S_n corresponds to the nondominated set. In order to obtain the efficient set with the MDP-BUKP algorithm, a binary string is maintained with each state and updated accordingly during the sequential process. For this reason, the implementation keeps states with the same component values. Only two sets of states are maintained during the overall sequential process since, at any stage $i > 0$, only set S_{i-1} is required. The code is written in C++ using the STL library.

For the connectedness analysis, the outcome of MDP-BUKP consists of the maximal complete efficient set. The 1-*flip* and the 1-*flip-exchange* neighborhood structures are considered for this problem. Gorski et al. (2011) theoretically show that the BUKP efficient set is not connected in the general case with respect to the 1-*flip* neighborhood operator. For each neighborhood, an adjacency matrix is built, indicating whether each two efficient solutions are neighbors or not. Based on this matrix, the connectedness of the corresponding efficient graph is tested. First, for those set of instances, the efficient set is always connected with respect to the 1-*flip-exchange* neighborhood. In addition, for each instance, we compute the proportion of connected solutions over the efficient

Table 2: Proportion of the largest connected component and number of unconnected solutions on the graph of efficient solutions of BUKP instances with respect to the 1-*flip* neighborhood (%larg and #miss, respectively). Average values, respectively rounded at 10^{-1} and 10^{0}, are reported.

	$\rho = -0.8$		$\rho = -0.4$		$\rho = 0.0$		$\rho = 0.4$		$\rho = 0.8$	
n	%larg	#miss	%larg	#miss	%larg	#miss	%larg	#miss	%larg	#miss
100	100.0	0	100.0	0	100.0	0	100.0	0	100.0	0
200	100.0	1	100.0	0	100.0	0	100.0	0	100.0	0
300	100.0	1	100.0	0	100.0	0	100.0	0	100.0	1
400	100.0	0	100.0	1	100.0	1	100.0	1	100.0	1
500	100.0	1	100.0	0	100.0	1	100.0	1	100.0	3
600	100.0	1	100.0	1	100.0	1	100.0	0	100.0	1
700	100.0	1	100.0	1	100.0	1	100.0	1	100.0	3
800	100.0	1	100.0	1	100.0	1	100.0	1	100.0	2
900	100.0	1	100.0	1	100.0	2	100.0	2	100.0	2
1000	100.0	2	100.0	3	100.0	1	100.0	2	100.0	2

set induced by the 1-*flip* neighborhood. Table 2 gives this average ratio of the largest connected component of the efficient graph, as well as the average number of solutions that are not connected to this largest connected component. Both numbers are rounded to the nearest value. Many instances are connected with respect to this smaller neighborhood structure. Moreover, although the correlation in the input data influences the size of the efficient set, more than 99.9% of efficient solutions belong to the same graph component for all the instances we investigated. As well, very few solutions are missing in average, *i.e.* at most 3 efficient solutions out of 89851 in the worst case ($n = 500$, $\rho = 0.8$).

3.3 Connectedness Analysis for BKP-BC

The efficient set of BKP-BC is computed by means of a multi-objective dynamic programming algorithm (MDP-BKP-BC), that extends the MDP-BUKP algorithm given in Section 3.2. This algorithm has $(k \cdot n)$ stages. At each stage, the algorithm generates the set $T_{(i,j)}$ of states, which represents a set of potential efficient solutions made up of the first i items, $i \in \{1, \ldots, n\}$, with cardinality j, $j \in \{1, \ldots, k\}$. A state $t = (t^p, t^w) \in T_{(i,j)}$ represents a feasible solution of profit t^p and weight t^w. This approach follows the recurrence relation:

$$T_{(i,j)} := \text{vmax}\left\{T_{(i-1,j)} \cup \left\{(t^s + p_i, t^w - w_i), t \in T_{(i-1,j-1)}\right\}\right\}$$

for $i \in \{1, \ldots, n\}$ and $j \in \{1, \ldots, k\}$, with the basis cases $T_{(i,0)} := \{(0,0)\}$, for $i \in \{0, \ldots, n\}$ and $T_{(0,j)} := \{(0,0)\}$ for $j \in \{0, \ldots, k\}$. The nondominated set of states is given by the set $\text{vmax}\left\{T_{(n,0)} \cup \cdots \cup T_{(n,k)}\right\}$. The implementation follows the same principles presented in Section 3.2 for the MDP-BUKP algorithm. However, the implementation of MDP-BKP-BC has to keep $2(k+1)$ sets during the overall process, since at each state $i \in \{1, \ldots, n\}$, the sets $T_{(i,j)}$ and $T_{(i-1,j)}$, for $j \in \{0, \ldots, k\}$ are required. For this reason, MDP-BKP-BC should take more time than MDP-BUKP for the same instance size.

First, let us note that MDP-BKP-BC did not find the efficient set for the following instances, due to an overload of RAM resources available: $k = n/2$, $\rho \in \{0.4, 0.8\}$, and $n \in \{800, 900, 1000\}$. In the same way than for the BUKP, we did not find any instance whose efficient set is not connected with respect to the 1-*flip-exchange* neighborhood. These results corroborate those of Gorski et al. (2011) for much smaller instances (up to 100 items). However, as reported in Table 3, and differently from the connectedness

Table 3: Proportion of the largest connected component and number of unconnected solutions on the graph of efficient solutions of BKP-BC instances with respect to the 1-*flip* neighborhood (%larg and #miss, respectively). Average values, respectively rounded at 10^{-1} and 10^{0}, are reported.

	$\rho = -0.8$		$\rho = -0.4$		$\rho = 0.0$		$\rho = 0.4$		$\rho = 0.8$	
n	%larg	#miss	%larg	#miss	%larg	#miss	%larg	#miss	%larg	#miss
					$k = n/10$					
100	93.2	23	73.6	57	66.6	120	57.7	150	58.2	286
200	90.4	81	72.5	214	58.9	320	55.0	457	59.5	908
300	89.7	157	71.9	447	58.8	650	55.1	853	60.2	1424
400	88.7	262	71.5	658	57.1	1060	54.8	1464	61.9	2565
500	88.9	346	71.4	1031	58.6	1698	55.3	2487	63.3	3236
600	89.1	369	71.8	1190	56.4	2104	54.9	3021	62.5	4978
700	88.5	555	72.1	1467	56.6	2662	55.4	4035	63.8	5569
800	89.7	557	72.7	1698	57.0	3551	55.5	4815	63.0	6842
900	89.1	767	72.1	2096	56.4	4095	54.8	5949	63.0	8252
1000	89.0	946	70.7	2757	55.8	5041	55.3	7537	63.7	10389
					$k = n/5$					
100	92.3	40	75.5	142	66.9	270	60.1	444	58.1	809
200	90.5	168	75.2	409	63.9	780	55.9	1125	54.1	2188
300	90.8	280	75.1	818	63.5	1608	56.1	2498	53.9	4043
400	89.8	640	74.0	1606	61.4	2685	54.1	3870	54.7	6671
500	90.9	672	75.7	2024	61.2	3992	54.1	5942	55.7	9673
600	91.1	965	75.9	2595	61.1	4755	54.3	7973	54.6	15112
700	90.1	1303	73.6	3194	62.7	6186	53.9	10022	55.3	18096
800	90.3	1337	74.5	4604	61.1	8373	53.3	12586	54.7	21088
900	90.6	1580	74.3	4902	61.2	8987	53.7	15817	54.0	26504
1000	90.3	2006	74.0	6007	61.3	11909	52.9	18678	55.0	29719
					$k = n/2$					
100	97.3	49	91.2	210	86.1	303	83.1	604	80.4	920
200	97.0	241	90.6	671	84.4	1111	80.5	2220	77.7	5125
300	96.7	390	90.5	1057	85.6	1914	80.2	3889	77.0	8059
400	96.5	552	89.9	1963	84.5	3751	80.6	6158	77.7	11725
500	96.6	846	90.4	2423	85.0	4946	79.7	9524	76.3	19235
600	96.6	1228	90.2	3466	84.5	6688	80.8	11692	78.0	24240
700	96.3	1411	89.5	4898	84.5	8917	79.5	17144	76.1	35859
800	96.4	1673	89.5	5813	84.0	12024	–	–	–	–
900	96.2	3489	89.4	9859	83.9	13635	–	–	–	–
1000	96.3	2950	89.7	8811	84.6	15489	–	–	–	–

results obtained for the first problem, no instance with an efficient set that is connected with respect to the 1-*flip* neighborhood was found. The size of the largest connected component of the efficient graph varies from more than 50% up to 97% of the cardinality of the efficient set. This proportion decreases with the increase of data correlation, and with the decrease of the cardinality bound.

3.4 Connectedness Analysis for BKP-FC

The multi-objective dynamic programming algorithm for BKP-FC (MDP-BKP-FC) is based on MDP-BKP-BC. The implementation follows the same principles. The only difference is on the computation of the nondominated set, which is now given by the set vmax $\left\{T_{(n,k)}\right\}$.

Given that all feasible solutions from BKP-FC must have the same number of items in the knapsack to satisfy the constraints, the 1-*exchange* neighborhood was considered for the connectedness analysis. Let us remark that MDP-BKP-FC was not able to solve the following large-size instances due to an overload of RAM resources available: $k = n/2$, $\rho = 0.8$ and $n \in \{900, 1000\}$. However, we did not find any unconnected case for all the other instances; see supplementary material for details (Liefooghe et al., 2012). This extends known results from Gorski (2010), based on the connectedness of the BKP-

Algorithm 1 Pseudo-code of the Pareto Local Search algorithm

Input: $n \in \mathbb{N}$
$p, w \in \mathbb{N}^n$
$x^0 \in X$
Output: Set V_T

1: $V_F := \{x^0\}$
2: $V_T := \emptyset$
3: **while** $V_F \neq \emptyset$ **do**
4: select $x^\star$ from V_F
5: $V_F := V_F \setminus \{x^\star\}$
6: $V_T := V_T \cup \{x^\star\}$
7: **for all** $x \in \mathcal{N}(x^\star)$ **do**
8: **if** $\{x' \mid x' \in V_F \cup V_T, (p(x'), -w(x')) \geq (p(x), -w(x))\} = \emptyset$ **then**
9: $V_F := \{x' \mid x' \in V_F, (p(x), -w(x)) \ngeq (p(x'), -w(x'))\}$
10: $V_T := \{x' \mid x' \in V_T, (p(x), -w(x)) \ngeq (p(x'), -w(x'))\}$
11: $V_F := V_F \cup \{x\}$
12: **end if**
13: **end for**
14: **end while**
15: **return** V_T

FC weakly efficient set and on Hamming distance 2.

4 Local Search for Bi-objective Knapsack Problems

This section reports an experimental analysis on the performance of a local search algorithm applied to several instances of the bi-objective knapsack problems under study.

4.1 Local Search Algorithm

The local search algorithm for all problems is based on the Pareto Local Search (PLS) (Paquete et al., 2007). The pseudo-code is given in Algorithm 1. For the sake of simplicity, let us assume that all feasible solutions have a distinct image in the objective space. Two archives of nondominated solutions are maintained, V_T and V_F, respectively. Archive V_T contains the set of solutions whose neighborhood has already been explored, while V_F contains the remaining ones. PLS starts with a feasible solution that initializes the archive. Then, at each iteration, a solution is chosen from V_F, and its neighborhood is explored ($\mathcal{N}(x)$ denotes the feasible neighbors of a given solution $x \in X$). All the nondominated neighboring solutions are used to update V_F and dominated solutions are discarded from V_F and V_T. The algorithm terminates once V_F is empty. This algorithm stops naturally when a *Pareto local optimum set* is found, it does not cycle, and if connectedness of the efficient set holds, it is able to identify a minimal complete set by starting with at least one efficient solution (Paquete et al., 2007).

In the following, some particular details of the implementation are described.

- *Selection.* At each iteration of the algorithm, the solution from V_F with the smallest profit value is selected (Algorithm 1, line 4).

- 1-*flip neighborhood exploration.* In order to perform the 1-*flip* neighborhood exploration in an efficient manner, a preliminary step, applied before the search process, ranks the items into different layers, with respect to the *dominance-depth ranking* (Goldberg, 1989). For instance, this ranking is performed between solutions in the NSGA-II algorithm (Deb et al., 2002). Two different ranks are required to cover

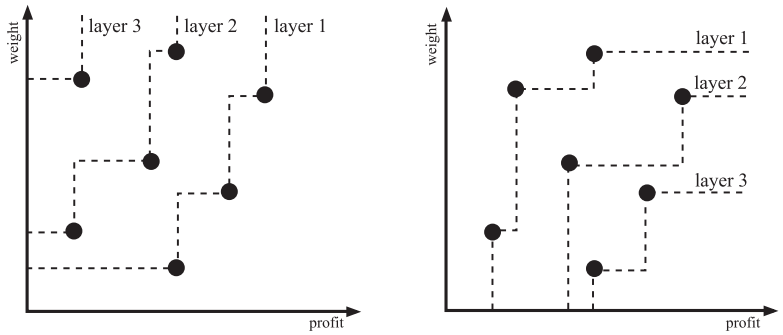


Figure 1: Dominance-depth ranking of items for the exploration of the neighborhood (left: ranking for adding, right: ranking for deleting).

the cases of adding and deleting an item. This ranking is illustrated in Fig. 1. Let L_k denote the set of items in the k-th layer and let $i \in L_k$ and $j \in L_l$, with $k < l$. Then, for the case of the addition, it holds that $(p_j, -w_j) \not\geq (p_i, -w_i)$. Let us consider an arbitrary solution $x^\star \in X$. The neighborhood exploration starts by examining the items in the first layer, and proceeds with the items in the subsequent layers. For each item i such that $x_i^\star = 0$, the corresponding neighboring solution x with $x_i = 1$ and $x_j = x_j^\star$ for $j \neq i$, is (incrementally) evaluated and is used to update the archive. Within a given layer, the exploration follows the nondecreasing order of the weight values. The exploration stops once no item of a given layer L_k belongs to the current solution, *i.e.* $\forall i \in L_k$, $x_i^\star = 0$. Indeed, it is not worth looking into subsequent layers because all the corresponding neighboring solutions are necessarily dominated by at least one neighboring solution built from L_k. For the case of deletion, a similar reasoning applies with the corresponding changes.

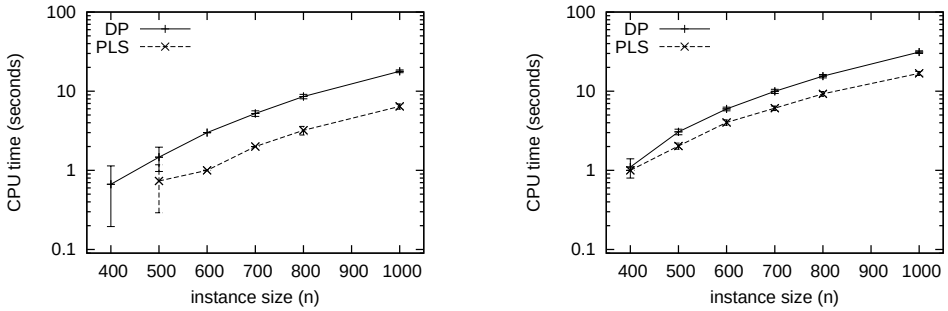
- 1-*exchange neighborhood exploration.* In order to exchange pairs of items more efficiently, the following pre-processing procedure is applied. First, for each item $i \in \{1, \ldots, n\}$, a set of $(n-1)$ tuples record the profit and weight difference with respect to every other item $j \neq i$. Hence, a tuple gives the difference, in terms of profit and weight values, of adding item i and removing item j in a knapsack. Then, these $(n-1)$ tuples are ranked in terms of dominance-depth for every item. When considering the exchange of item i with another item, the exploration follows the order given by the dominance-depth, with respect to the nondecreasing order of the weights for items within the same layer. The exploration stops when no item of a given layer belongs to the current solution. The exploration is iterated for every item inside the current solution.

- 1-*flip-exchange neighborhood exploration.* For this neighborhood, the same reasoning as above applies when adding or removing an item, and when exchanging two items. Three steps are then performed during the pre-processing phase.

- *Data structures.* Archives V_F and V_T are implemented as red-black trees. The removal of dominated solutions follows the algorithm of Kung et al. (1975).

Table 4: Proportion of efficient solutions, number of missing solutions, and I_ϵ-value of efficient set approximations found by PLS–*flip* for BUKP instances (%eff, #miss and I_ϵ, respectively). Average values, rounded at 10^{-1}, are reported.

	$\rho = -0.8$			$\rho = -0.4$			$\rho = 0.0$			$\rho = 0.4$			$\rho = 0.8$		
n	%eff	#miss	I_ϵ	%eff	#miss	I_ϵ	%eff	#miss	I_ϵ	%eff	#miss	I_ϵ	%eff	#miss	I_ϵ
100	1.0	0.0	1.0	1.0	0.0	1.0	1.0	0.0	1.0	1.0	0.0	1.0	1.0	0.0	1.0
200	1.0	0.0	1.0	1.0	0.0	1.0	1.0	0.0	1.0	1.0	0.0	1.0	1.0	0.0	1.0
300	1.0	0.0	1.0	1.0	0.0	1.0	1.0	0.0	1.0	1.0	0.0	1.0	1.0	0.2	1.0
400	1.0	0.0	1.0	1.0	0.0	1.0	1.0	0.1	1.0	1.0	0.1	1.0	1.0	0.0	1.0
500	1.0	0.1	1.0	1.0	0.0	1.0	1.0	0.1	1.0	1.0	0.1	1.0	1.0	0.1	1.0
600	1.0	0.1	1.0	1.0	0.1	1.0	1.0	0.0	1.0	1.0	0.0	1.0	1.0	0.1	1.0
700	1.0	0.1	1.0	1.0	0.0	1.0	1.0	0.0	1.0	1.0	0.1	1.0	1.0	0.2	1.0
800	1.0	0.2	1.0	1.0	0.1	1.0	1.0	0.1	1.0	1.0	0.1	1.0	1.0	0.3	1.0
900	1.0	0.2	1.0	1.0	0.0	1.0	1.0	0.1	1.0	1.0	0.3	1.0	1.0	0.2	1.0
1000	1.0	0.2	1.0	1.0	0.1	1.0	1.0	0.1	1.0	1.0	0.3	1.0	1.0	0.3	1.0



(a) $\rho = -0.4$ (b) $\rho = 0.4$

Figure 2: CPU time in seconds (average value and deviation, given in log-scale) of MDP-BUKP and PLS–*flip* for BUKP instances. Values below 0.1 seconds are not reported to increase readability.

4.2 Experimental Analysis for BUKP

The dynamic programming algorithm MDP-BUKP and two PLS versions, based on the 1-*flip* and the 1-*flip-exchange* neighborhood structures, respectively denoted by PLS–*flip* and PLS–*flip-exchange*, were run on the BUKP instances described in Section 2.5. The PLS algorithms all start with the same initial efficient solution $x^0 = \mathbf{0}$. This solution maps, on the objective space, to an extreme point from the nondominated set, with null profit and weight values. Our hope is that the local search algorithms will perform efficiently due to the connectedness results reported in Section 3.2.

The implementation of MDP-BUKP was modified so that the outcome consists of a minimal complete efficient set. All the algorithms share the same programming language, data structures, compilation options and were run in the same machine. The experiments were conducted on an Intel Core 2 Quad processor with 2.40 GHZ and 4GB RAM under Ubuntu 10.04. All codes were compiled with g++ version 4.4.3 using the -O3 flag. The CPU-time taken by PLS–*flip* and MDP-BUKP for $\rho = -0.4$ and $\rho = 0.4$ is reported in Fig. 2. Similar relationship between performance of both implementations was obtained for the remaining correlation values (Liefooghe et al., 2012). Since PLS–*flip-exchange* took always significantly more time than MDP-BUKP, the results of the former are omitted. Table 4 presents the percentage of efficient solutions and the number of missing solutions returned by PLS–*flip*, averaged over the results obtained

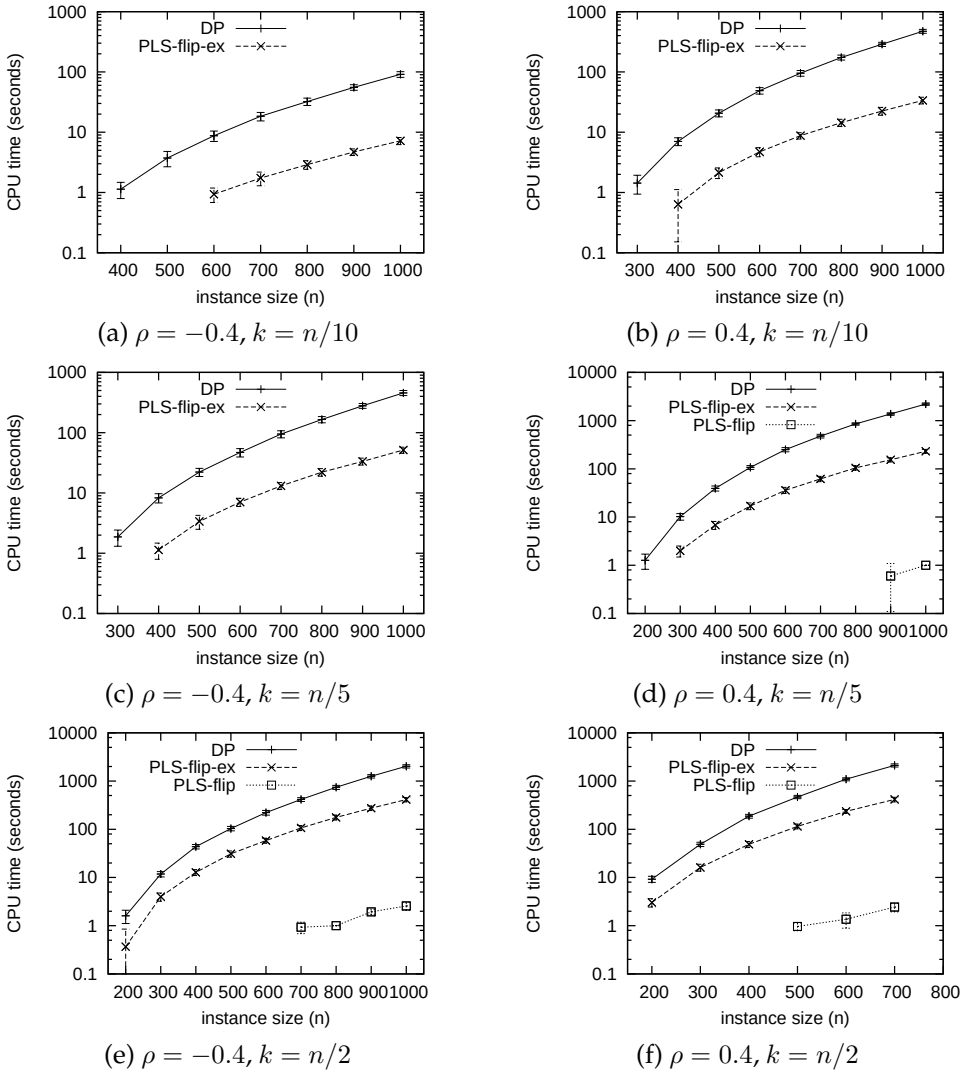
Table 5: Proportion of efficient solutions, number of missing solutions, and I_ϵ-value of efficient set approximations found by PLS–*flip* for BKP-BC instances (%eff, #miss and I_ϵ, respectively). Average values, rounded at 10^{-1}, are reported.

	$\rho = -0.8$			$\rho = -0.4$			$\rho = 0.0$			$\rho = 0.4$			$\rho = 0.8$		
n	%eff	#miss	I_ϵ	%eff	#miss	I_ϵ	%eff	#miss	I_ϵ	%eff	#miss	I_ϵ	%eff	#miss	I_ϵ
							$k = n/10$								
100	0.9	6.2	1.0	0.7	32.7	1.1	0.7	55.9	1.1	0.5	102.0	1.2	0.5	225.4	1.3
200	0.9	30.0	1.0	0.7	120.7	1.1	0.6	213.1	1.2	0.5	388.6	1.3	0.4	832.3	1.4
300	0.9	65.2	1.0	0.7	246.3	1.1	0.6	458.0	1.2	0.5	783.2	1.3	0.4	1641.3	1.4
400	0.9	128.1	1.0	0.7	408.0	1.1	0.6	778.0	1.2	0.5	1345.3	1.3	0.4	2876.6	1.4
500	0.9	190.0	1.0	0.7	630.2	1.1	0.6	1139.7	1.2	0.5	2078.9	1.3	0.4	4347.4	1.4
600	0.9	253.8	1.0	0.7	821.2	1.1	0.6	1612.1	1.2	0.5	2905.4	1.3	0.4	5936.8	1.4
700	0.9	354.5	1.0	0.7	1079.9	1.1	0.6	2112.3	1.2	0.5	3801.9	1.3	0.4	7855.3	1.4
800	0.9	409.4	1.0	0.7	1292.2	1.1	0.6	2610.0	1.2	0.5	4673.6	1.3	0.4	9421.1	1.4
900	0.9	525.1	1.0	0.7	1638.8	1.1	0.6	3254.1	1.2	0.5	5750.2	1.3	0.4	11619.5	1.4
1000	0.9	648.8	1.0	0.7	2076.4	1.1	0.6	3937.2	1.2	0.5	7001.7	1.3	0.4	14048.7	1.4
							$k = n/5$								
100	0.9	18.3	1.0	0.8	80.2	1.1	0.7	137.1	1.1	0.6	257.5	1.2	0.5	524.3	1.3
200	0.9	81.7	1.0	0.8	268.2	1.1	0.6	509.7	1.2	0.6	859.9	1.2	0.5	1826.4	1.3
300	0.9	159.7	1.0	0.8	540.1	1.1	0.6	1005.8	1.2	0.6	1798.5	1.2	0.5	3730.6	1.3
400	0.9	296.7	1.0	0.7	953.7	1.1	0.6	1875.2	1.2	0.5	3177.1	1.3	0.5	6401.9	1.3
500	0.9	403.7	1.0	0.8	1321.7	1.1	0.6	2746.7	1.2	0.5	4870.5	1.3	0.5	9895.4	1.4
600	0.9	543.4	1.0	0.8	1796.9	1.1	0.6	3781.7	1.2	0.5	6768.4	1.3	0.5	13245.3	1.4
700	0.9	784.8	1.0	0.7	2593.5	1.1	0.6	4718.2	1.2	0.5	8600.6	1.3	0.5	16922.8	1.4
800	0.9	972.2	1.0	0.7	3086.2	1.1	0.6	6190.5	1.2	0.5	10647.3	1.3	0.5	20682.9	1.4
900	0.9	1155.9	1.0	0.7	3839.8	1.1	0.6	7539.1	1.2	0.5	12832.6	1.3	0.5	24831.5	1.4
1000	0.9	1435.4	1.0	0.7	4670.4	1.1	0.6	8983.7	1.2	0.5	15663.6	1.3	0.5	30061.9	1.4
							$k = n/2$								
100	1.0	21.7	1.0	0.9	91.6	1.0	0.9	180.6	1.1	0.8	286.3	1.1	0.8	576.3	1.1
200	1.0	86.0	1.0	0.9	325.9	1.0	0.8	717.6	1.1	0.8	1192.8	1.1	0.8	2352.9	1.1
300	1.0	191.3	1.0	0.9	680.6	1.0	0.9	1281.2	1.1	0.8	2484.1	1.1	0.8	4942.1	1.1
400	1.0	340.4	1.0	0.9	1205.3	1.0	0.8	2381.4	1.1	0.8	4046.7	1.1	0.8	8019.2	1.1
500	1.0	490.9	1.0	0.9	1704.1	1.0	0.9	3347.7	1.1	0.8	6357.4	1.1	0.8	12289.5	1.2
600	1.0	683.1	1.0	0.9	2319.6	1.0	0.8	4839.5	1.1	0.8	8203.4	1.1	0.8	15702.7	1.2
700	1.0	970.6	1.0	0.9	3262.5	1.1	0.8	6299.8	1.1	0.8	11395.7	1.1	0.8	21414.3	1.2
800	1.0	1194.9	1.0	0.9	4142.0	1.0	0.8	8090.1	1.1	–	–	–	–	–	–
900	1.0	1542.9	1.0	0.9	5061.5	1.0	0.8	9901.3	1.1	–	–	–	–	–	–
1000	1.0	1788.0	1.0	0.9	5915.4	1.1	0.8	11444.0	1.1	–	–	–	–	–	–

in 30 instances for each size and correlation. Moreover, we report statistics related to the the multiplicative ϵ-indicator (I_ϵ) proposed by Zitzler et al. (2003). It gives the minimum multiplicative factor by which an approximation set A has to be translated in the objective space in order to weakly dominate the efficient set $X_E^\star$. In other words, $I_\epsilon(A) = 1$ means that the approximation found coincides with the nondominated set. I_ϵ can be defined as follows.

$$I_\epsilon(A) = \min_{\epsilon \in \mathbb{R}^+} \{\forall x \in X_E^\star, \exists x' \in A : p(x') \cdot \epsilon \geqslant p(x), w(x') \cdot \epsilon \leqslant w(x)\} \quad (1)$$

Although the experimental analysis indicated the existence of instances with unconnected efficient set with respect to the 1-*flip* neighborhood (see Table 2), the results show that the local search approach using the same notion of neighborhood is able to identify a minimal complete set in many cases. For other instances, it leads to the identification of more than 99.9% of the efficient set, within I_ϵ-values very close to 1. The number of missing efficient solutions is then negligible compared to the cardinality of the efficient set. Furthermore, the local search performs very efficiently in comparison to the dynamic programming approach in terms of computational time. Indeed, the larger the instance size, the larger the gap between PLS–*flip* and MDP-BUKP in terms of CPU time. However, PLS–*flip* appears to be slightly more efficient for negatively correlated data.



(a) $\rho = -0.4$, $k = n/10$ (b) $\rho = 0.4$, $k = n/10$

(c) $\rho = -0.4$, $k = n/5$ (d) $\rho = 0.4$, $k = n/5$

(e) $\rho = -0.4$, $k = n/2$ (f) $\rho = 0.4$, $k = n/2$

Figure 3: CPU time in seconds (average value and deviation, given in $\log$-scale) of MDP-BKP-BC, PLS–*flip* and PLS–*flip-exchange* for BKP-BC instances. Values below 0.1 seconds are not reported to increase readability.

4.3 Experimental Analysis for BKP-BC

Given the positive results reported in the previous section, a similar local search algorithm was developed under the same reasoning as for BUKP. The same initial solution, $x^0 = \mathbf{0}$, is used to start the search process. The only difference is in the neighborhood exploration, since a maximum number of items has to be ensured in the solution when considering the possibility of adding an item, *i.e.* only feasible solutions are considered.

MDP-BKP-BC and the two versions of PLS were run in the BKP-BC instances. The experiments were performed on the same machine, with g++ version 4.4.3 using the -O3 flag. As for the connectedness analysis, MDP-BKP-BC was not able to terminate

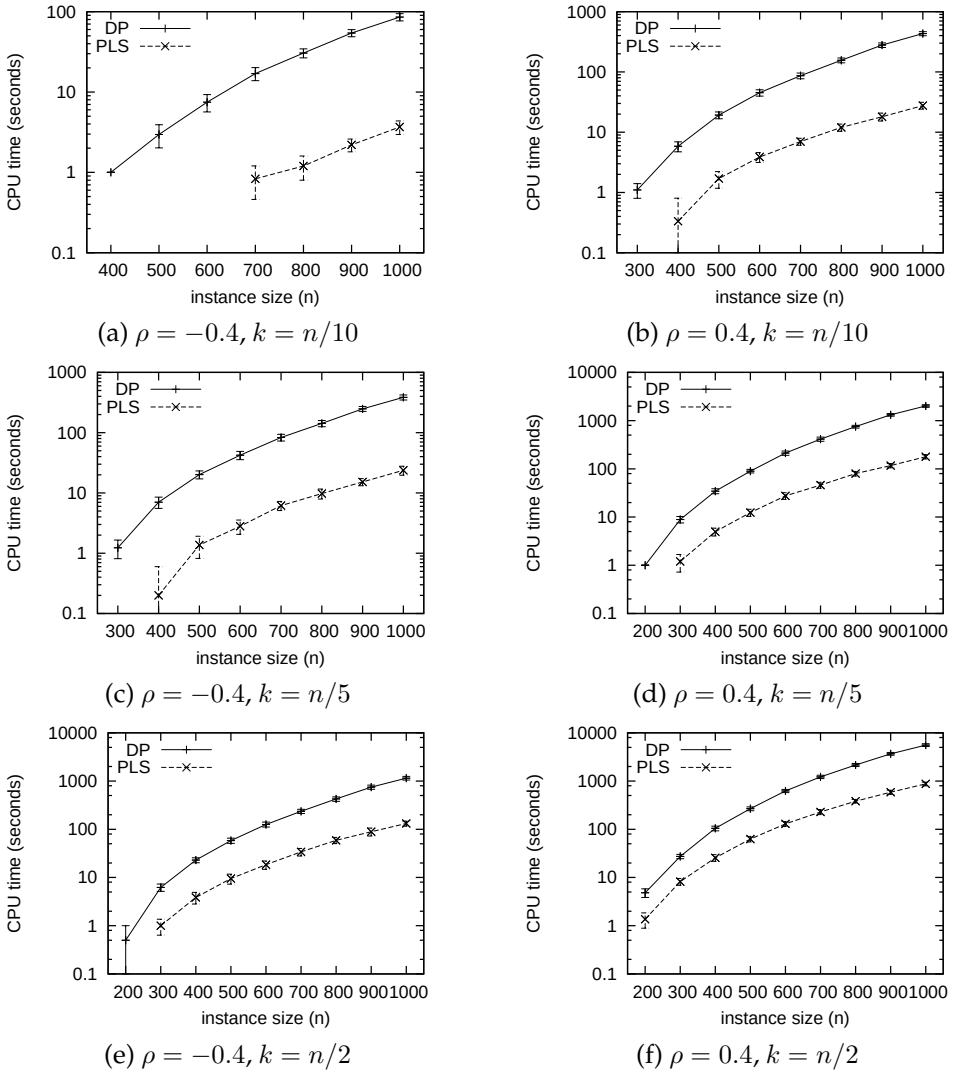
Table 6: Proportion of efficient solutions, number of missing solutions, and I_ϵ-value of efficient set approximations found by PLS–*exchange* for BKP-FC instances (%eff, #miss and I_ϵ, respectively). Average values, rounded at 10^{-1}, are reported.

	$\rho = -0.8$			$\rho = -0.4$			$\rho = 0.0$			$\rho = 0.4$			$\rho = 0.8$		
n	%eff	#miss	I_ϵ	%eff	#miss	I_ϵ	%eff	#miss	I_ϵ	%eff	#miss	I_ϵ	%eff	#miss	I_ϵ
							$k = n/10$								
100	1.0	0.0	1.0	1.0	0.0	1.0	1.0	0.0	1.0	1.0	0.0	1.0	1.0	0.0	1.0
200	1.0	0.0	1.0	1.0	0.0	1.0	1.0	0.0	1.0	1.0	0.0	1.0	1.0	0.0	1.0
300	1.0	0.0	1.0	1.0	0.0	1.0	1.0	0.0	1.0	1.0	0.0	1.0	1.0	0.0	1.0
400	1.0	0.0	1.0	1.0	0.0	1.0	1.0	0.0	1.0	1.0	0.0	1.0	1.0	0.0	1.0
500	1.0	0.0	1.0	1.0	0.0	1.0	1.0	0.0	1.0	1.0	0.0	1.0	1.0	0.0	1.0
600	1.0	0.0	1.0	1.0	0.0	1.0	1.0	0.0	1.0	1.0	0.0	1.0	1.0	0.0	1.0
700	1.0	0.0	1.0	1.0	0.0	1.0	1.0	0.0	1.0	1.0	0.0	1.0	1.0	0.0	1.0
800	1.0	0.0	1.0	1.0	0.0	1.0	1.0	0.0	1.0	1.0	0.0	1.0	1.0	0.0	1.0
900	1.0	0.0	1.0	1.0	0.0	1.0	1.0	0.0	1.0	1.0	0.0	1.0	1.0	0.0	1.0
1000	1.0	0.0	1.0	1.0	0.0	1.0	1.0	0.0	1.0	1.0	0.0	1.0	1.0	0.0	1.0
							$k = n/5$								
100	1.0	0.0	1.0	1.0	0.0	1.0	1.0	0.0	1.0	1.0	0.0	1.0	1.0	0.0	1.0
200	1.0	0.0	1.0	1.0	0.0	1.0	1.0	0.0	1.0	1.0	0.0	1.0	1.0	0.0	1.0
300	1.0	0.0	1.0	1.0	0.0	1.0	1.0	0.0	1.0	1.0	0.0	1.0	1.0	0.0	1.0
400	1.0	0.0	1.0	1.0	0.0	1.0	1.0	0.0	1.0	1.0	0.0	1.0	1.0	0.0	1.0
500	1.0	0.0	1.0	1.0	0.0	1.0	1.0	0.0	1.0	1.0	0.0	1.0	1.0	0.0	1.0
600	1.0	0.0	1.0	1.0	0.0	1.0	1.0	0.0	1.0	1.0	0.0	1.0	1.0	0.0	1.0
700	1.0	0.0	1.0	1.0	0.0	1.0	1.0	0.0	1.0	1.0	0.0	1.0	1.0	0.0	1.0
800	1.0	0.0	1.0	1.0	0.0	1.0	1.0	0.0	1.0	1.0	0.0	1.0	1.0	0.0	1.0
900	1.0	0.0	1.0	1.0	0.0	1.0	1.0	0.0	1.0	1.0	0.0	1.0	1.0	0.0	1.0
1000	1.0	0.0	1.0	1.0	0.0	1.0	1.0	0.0	1.0	1.0	0.0	1.0	1.0	0.0	1.0
							$k = n/2$								
100	1.0	0.0	1.0	1.0	0.0	1.0	1.0	0.0	1.0	1.0	0.0	1.0	1.0	0.0	1.0
200	1.0	0.0	1.0	1.0	0.0	1.0	1.0	0.0	1.0	1.0	0.0	1.0	1.0	0.0	1.0
300	1.0	0.0	1.0	1.0	0.0	1.0	1.0	0.0	1.0	1.0	0.0	1.0	1.0	0.0	1.0
400	1.0	0.0	1.0	1.0	0.0	1.0	1.0	0.0	1.0	1.0	0.0	1.0	1.0	0.0	1.0
500	1.0	0.0	1.0	1.0	0.0	1.0	1.0	0.0	1.0	1.0	0.0	1.0	1.0	0.0	1.0
600	1.0	0.0	1.0	1.0	0.0	1.0	1.0	0.0	1.0	1.0	0.0	1.0	1.0	0.0	1.0
700	1.0	0.0	1.0	1.0	0.0	1.0	1.0	0.0	1.0	1.0	0.0	1.0	1.0	0.0	1.0
800	1.0	0.0	1.0	1.0	0.0	1.0	1.0	0.0	1.0	1.0	0.0	1.0	1.0	0.0	1.0

due to an overload of RAM resources available. Indeed, for $k = n/2$ and $\rho = 0.8$, the maximum number of states maintained by the algorithm is $10.166.945$ in average for $n = 700$, whereas it goes higher than $11.000.000$ states after two hours of CPU time for all the BKP-BC instances under consideration. In contrast, the maximum number of solutions (the archive size) maintained by PLS-BKP-BC is 105.811 in average for $n = 700$, and 125.323 in average for $n = 800$. The CPU-time taken by the three approaches is plotted in Fig. 3 for six different instance parameter settings. Clearly, both PLS versions take much less time than MDP-BKP-BC to terminate. Similar results hold for the remaining instances; see supplementary material for details (Liefooghe et al., 2012). Moreover, as expected from the connectedness analysis reported in the previous section, PLS–*flip-exchange* was always able to find a minimal complete set for all instances up to 1000 items in less than one hour of CPU-time. Clearly, PLS–*flip-exchange* takes advantage of the connectedness property reported in Section 3.3. Table 5 reports quality metrics for the solutions found by PLS–*flip*. PLS–*flip* is, in many cases, able to find more than half of a minimal complete set in less than one second, except for $\rho = 0.8$ and $k = n/10$, where only 40% is found. Furthermore, the performance of PLS–*flip* decreases as the cardinality bound decreases, and as the data correlation increases. The problem size has a low influence on the results.

4.4 Experimental Analysis for BKP-FC

For BKP-FC, we consider a PLS algorithm based on the 1-*exchange* neighborhood, denoted by PLS–*exchange*. As for BKP-BC, only feasible neighboring solutions are considered as candidates to update the archive. The following efficient solution is considered



(a) $\rho = -0.4$, $k = n/10$ (b) $\rho = 0.4$, $k = n/10$

(c) $\rho = -0.4$, $k = n/5$ (d) $\rho = 0.4$, $k = n/5$

(e) $\rho = -0.4$, $k = n/2$ (f) $\rho = 0.4$, $k = n/2$

Figure 4: CPU time in seconds (average value and deviation, given in $\log$-scale) of MDP-BKP-FC and PLS–*exchange* for BKP-FC instances. Values below 0.1 seconds are not reported to increase readability.

to initialize the search process. We sort all the items with respect to weight values in the increasing order, and the k first items are included in the initial solution. As a consequence, the solution maps to an extreme point of the nondominated set, with lowest weight and profit values with respect to the cardinality k under consideration.

MDP-BKP-FC and PLS–*exchange* were run on the BKP-FC instances described in Section 2.5. The experiments were performed on the same machine, with the same compilation options. Fig. 4 gives the CPU time taken by both algorithms to terminate for $\rho \in \{-0.4, 0.4\}$. For other ρ-values, please refer to supplementary materials (Liefooghe et al., 2012). Table 6 gives qualitative values of solutions found by PLS–*exchange*. A

minimal complete efficient set was always found by PLS–*exchange* for all the instances we experimented in less than one hour. The higher the cardinality constraint, the higher the CPU time taken by both algorithms to terminate. However, PLS–*exchange* is always taking much less time than MDP-BKP-FC. As expected from the results reported in Section 3.4, PLS–*exchange* is able to outperform MDP-BKP-FC by exploiting the connectedness of efficient solutions.

5 Concluding Remarks

This article describes an experimental analysis on the structure of the efficient set, in terms of connectedness, for three MoCO problems. Even if theoretical investigations for similar problems report unconnected efficient set in the general case (Ehrgott and Klamroth, 1997; Gorski et al., 2011), the experimental analysis for the problems investigated in this paper are quite promising. For all bi-objective versions of the unconstrained knapsack problem and knapsack problem with a bounded and a fixed cardinality constraint, the experiments suggest that small-sized neighborhood structures give rise to connected efficient sets quite frequently, and independently of the size and of the structure of input data. In fact, it is not yet clear what structure of the input data may generate, in general, an unconnected efficient set under the 1-*flip-exchange* neighborhood that was used in this article.

Although the large number of connected instances motivates the use of local search algorithms, it is still an open question whether those approaches are efficient enough as compared to exact algorithms. The experimental analysis reported in this article gives a clear positive answer for problems with a cardinality constraint. For the first problem, without cardinality constraint, some preliminary results indicated that the local search proposed in this article under the same neighborhood that (empirically) provides connectedness would not be worthwhile in terms of running time. Still, using a smaller neighborhood structure allows the same algorithm to find more than 99.9% of the efficient set in a significantly less amount of time than the exact approach.

The simplicity of the local search proposed in this article is very appealing: it needs no definition of parameters and requires a minimum number of modifications in order to be applied to other type of knapsack problems, even with more than two objective functions. For instance, the same principles can be applied to the multi-objective knapsack problem (with several maximizing profit objectives and one capacity constraint) by ignoring or penalizing infeasible neighboring solutions. However, finding appropriate definitions of neighborhoods that give rise to large number of connected efficient sets for knapsack problems with capacity constraints is still under investigation.

A natural question is whether it is possible to derive analytical results for MoCO problems that would prove connectedness by assuming some structure or distribution on the input data. Connections with neighborhood structures arising in the context of the linear programming formulation of the MoCO problem may provide further insights (Gorski et al., 2011). Moreover, the derivation of bounds on the run-time of multi-objective evolutionary algorithms that start from efficient solutions are also of interest. For instance, the size of the efficient set for the first problem is polynomially bounded for many input data distributions (Beier and Vöcking, 2004). This suggests that a polynomial run-time bound may be achieved by such type of algorithms.

Acknowledgments. The authors are grateful to Marco Simões for his help on a preliminary version of this work. They acknowledge Jochen Gorski for the discussions on the main topic of this article. Thanks are also due to three anonymous referees for their valuable and constructive comments. This work was partially supported by the Portuguese Foundation for Science and

Technology (PTDC/EIA-CCO/098674/2008) and the project "Connectedness and Local Search for Multiobjective Combinatorial Optimization" funded by the Deutscher Akademischer Austausch Dienst and Conselho de Reitores das Universidades Portuguesas.

References

Beier, R. and Vöcking, B. (2004). Probabilistic analysis of knapsack core algorithms. In *Proceedings of the 15th ACM-SIAM Symposium on Discrete Algorithms*, pages 468–477. Society for Industrial and Applied Mathematics.

Deb, K., Agrawal, S., Pratap, A., and Meyarivan, T. (2002). A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197.

Dubois-Lacoste, J., López-Ibáñez, M., and Stützle, T. (2011). A hybrid TP+PLS algorithm for bi-objective flow-shop scheduling problems. *Computers & OR*, 38(8):1219–1236.

Ehrgott, M. (2000). *Multicriteria optimization*, volume 491 of *Lecture Notes in Economics and Mathematical Systems*. Springer, Berlin, Germany.

Ehrgott, M. and Klamroth, K. (1997). Connectedness of efficient solutions in multiple criteria combinatorial optimization. *European Journal of Operational Research*, 97(1):159–166.

Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, Boston, MA, USA.

Gomes da Silva, C., Clímaco, J., and Figueira, J. R. (2004). Geometrical configuration of the Pareto frontier of the bi-criteria 0-1-knapsack problem. Technical Report 16/2004, INESC, Coimbra, Portugal.

Gorski, J. (2010). *Multiple Objective Optimization and Implications for Single Objective Optimization*. PhD thesis, Bergische Universitüt Wuppertal.

Gorski, J., Klamroth, K., and Ruzika, S. (2011). Connectedness of efficient solutions in multiple objective combinatorial optimization. *Journal of Optimization Theory and Applications*, 150(3):475–497.

Gorski, J., Paquete, L., and Pedrosa, F. (2012). Greedy algorithms for a class of knapsack problems with binary weights. *Computers & Operations Research*, 39(3):498–511.

Knowles, J. D. and Corne, D. (2000). Approximating the nondominated front using the Pareto archived evolution strategy. *Evolutionary Computation*, 8(2):149–172.

Kung, H., Luccio, F., and Preparata, F. (1975). On finding the maxima of a set of vectors. *Journal of ACM*, 22(4):469–476.

Laumanns, M., Thiele, L., and Zitzler, E. (2004). Running time analysis of evolutionary algorithms on a simplified multiobjective knapsack problem. *Natural Computing*, 3(1):37–51.

Liefooghe, A., Humeau, J., Mesmoudi, S., Jourdan, L., and Talbi, E.-G. (2011a). On dominance-based multiobjective local search: design, implementation and experimental analysis on scheduling and traveling salesman problems. *Journal of Heuristics*. To appear.

Liefooghe, A., Paquete, L., and Figueira, J. R. (2012). Supplementary material: On local search for bi-objective knapsack problems. http://sites.google.com/site/arnaudliefooghe/sup/knapsack/.

Liefooghe, A., Paquete, L., Simoes, M., and Figueira, J. R. (2011b). Connectedness and local search for bicriteria knapsack problems. In *Proceedings of the 11th European Conference on Evolutionary Computation in Combinatorial Optimization (EvoCOP 2011)*, volume 6622 of *Lecture Notes in Computer Science*, pages 48–59. Springer.

Lust, T. and Teghem, J. (2010). Two-phase Pareto local search for the biobjective traveling salesman problem. *Journal of Heuristics*, 16(3):475–510.

Nemhauser, G. L. and Ullman, Z. (1969). Discrete dynamic programming and capital allocation. *Management Science*, 15(9):494–505.

Paquete, L., Schiavinotto, T., and Stützle, T. (2007). On local optima in multiobjective combinatorial optimization problems. *Annals of Operations Research*, 156(1):83–97.

Paquete, L. and Stützle, T. (2009). Clusters of non-dominated solutions in multiobjective combinatorial optimization: An experimental analysis. In *Multiobjective Programming and Goal Programming: Theoretical Results and Practical Applications*, volume 618 of *Lecture Notes in Economics and Mathematical Systems*, pages 69–77. Springer.

Verel, S., Liefooghe, A., Jourdan, L., and Dhaenens, C. (2011). Analyzing the effect of objective correlation on the efficient set of MNK-landscapes. In *Proceedings of the 5th Conference on Learning and Intelligent OptimizatioN (LION 5)*, volume 6683 of *Lecture Notes in Computer Science*, pages 116–130. Springer.

Zitzler, E., Thiele, L., Laumanns, M., Fonseca, C. M., and Grunert da Fonseca, V. (2003). Performance assessment of multiobjective optimizers: An analysis and review. *IEEE Transactions on Evolutionary Computation*, 7(2):117–132.