Connectedness and Local Search for Bicriteria Knapsack Problems

Arnaud Liefooghe, Luis Paquete, Marco Simoes, José Figueira

Connectedness and Local Search for Bicriteria Knapsack Problems

Arnaud Liefooghe[1], Luís Paquete[2], Marco Simões[2], and José R. Figueira[3]

[1] Université Lille 1, LIFL – CNRS – INRIA Lille-Nord Europe, France
arnaud.liefooghe@univ-lille1.fr
[2] CISUC, Department of Informatics Engineering, University of Coimbra, Portugal
paquete@dei.uc.pt,msimoes@student.dei.uc.pt
[3] INPL, École des Mines de Nancy, Laboratoire LORIA, France
jose.figueira@mines.inpl-nancy.fr

Abstract. This article reports an experimental study on a given structural property of connectedness of optimal solutions for two variants of the bicriteria knapsack problem. A local search algorithm that explores this property is then proposed and its performance is compared against exact algorithms in terms of running time and number of optimal solutions found. 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 approaches.

1 Introduction

Stochastic local search algorithms have been applied successfully to many multicriteria combinatorial optimization (MCO) problems. It is widely accepted that their sound performance is related to some structural properties of the solution 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 this class of algorithms.

In this article, the notion of *connectedness* of the set of optimal solutions for MCO problems (a.k.a. *the efficient set*) [3] is analyzed from an experimental point of view, and related to the performance of a particular class of multicriteria local search algorithms [9]. For a given efficient set of a MCO problem instance, a graph can be constructed such that each node represents an optimal solution and an edge connects two nodes if the corresponding optimal 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 [9], by starting from at least one efficient solution. However, worst-case results have shown that the efficient set for many MCO problems is not connected in general with respect to different neighborhood structures [3, 5], except for very few particular cases [6, 11]. Moreover, some recent results indicate that

approximate solutions that are obtained from independent metaheuristic runs on the bicriteria traveling salesman problem are strongly clustered with respect to small-sized neighborhood structures [10].

The degree of connectedness is here investigated experimentally for two variants of the bicriteria knapsack problem: The bicriteria unconstrained optimization problem (BUOP) and the bicriteria knapsack problem with bounded cardinality (BKP-BC). Both problems are NP-hard and intractable in the general case [2]. The experimental results suggest that the efficient set for the two problems above is very often connected with respect to elementary neighborhood structures, despite of the negative results reported in the literature [5]. Based on these positive findings, a local search algorithm is proposed and its performance is compared with that of multicriteria dynamic programming algorithms in terms of running-time and number of optimal 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. Section 2 and Section 3 give the connectedness results for BUOP and BKP-BC, respectively. Moreover, for each of the sections above, the local search algorithms and the exact approaches for the corresponding problem are introduced. Numerical results are shown on a large set of random instances of different structure and size. Finally, Section 4 presents conclusions and further work.

2 The Bicriteria Unconstrained Optimization Problem

This section introduces a variant of the classical knapsack problem where the capacity constraint is transformed into an additional criterion to be optimized. Then, an experimental analysis on the connectedness property of the corresponding efficient set is reported, as well as the performance of a local search algorithm on several instances of the problem.

2.1 Problem Definition

The original (single-criterion) 0/1 knapsack problem is formulated as follows:

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

where $p = (p_1, p_2, \ldots, p_j, \ldots, p_n)$ is the profit vector, p_j representing the amount of profit on item j, $j = 1, \ldots, n$, and $x = (x_1, x_2, \ldots, x_j, \ldots, x_n)$ with $x_j = 1$ if the item j is included in the subset of selected items (knapsack) and $x_j = 0$ otherwise; $w = (w_1, w_2, \ldots, w_j, \ldots, w_n)$ is the weight vector, w_j representing the amount of investment on item j, $j = 1, \ldots, n$; 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.

By transforming the capacity constraint of Problem (1) into a criterion, the following bicriteria unconstrained optimization problem [2] is obtained:

$$\max\ (p(x), -w(x))\,. \tag{2}$$

A proper meaning to the operator "max" above is given as follows. Let X denote the set of feasible solutions of Problem (2). The image of the feasible solutions when using the vector maximizing function of Problem (2) defines the feasible region in the criteria 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 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. The usual goal of MCO is to find *a minimal complete set*, that is, the smallest subset of the efficient set whose image coincides with the nondominated set. This subset may not be unique.

2.2 Connectedness Analysis

This section describes an experimental analysis for investigating the influence of problem size and degree of conflict between the two criteria on the connectedness property of the efficient set for BUOP. A multicriteria dynamic programming (MDP-BUOP) algorithm is implemented to compute the efficient set. This algorithm consists of the first phase of the Nemhauser-Ullman algorithm for the single-criterion 0/1 knapsack problem [8]. It has shown to be theoretically efficient for several input data distributions [1]. The MDP-BUOP sequential process consists of n stages. At any stage i, the algorithm generates a set S_i of states, which represents a set of *promising* feasible solutions made up of the first i items, $i = 1, \ldots, n$. A state $s = (s^p, s^w) \in S_i$ represents a feasible solution of profits s^p and weight s^w. The MDP-BUOP algorithm follows the recursion:

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

for $i = 1, \ldots, n$, with the basis case $S_0 := (0, 0)$. Operator "vmax" returns the states that are 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-BUOP algorithm, a binary string is generated with each new state and updated accordingly during the sequential process. For this reason, the implementation keeps states with the same component values. The removal of dominated states at each stage is performed by the algorithm of Kung et al. [7]. 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.

Table 1. Percentage of BUOP instances connected with respect to the 1-*flip* neighborhood (all instances are connected with respect to the 1-*flip*-*exchange* neighborhood) and average size of the efficient set for each instance size and data correlation (%*con* and *avgs*, respectively).

	$\rho = -0.8$		$\rho = -0.4$		$\rho = 0.0$		$\rho = 0.4$		$\rho = 0.8$	
size	%*con*	*avgs*	%*con*	*avgs*	%*con*	*avgs*	%*con*	*avgs*	%*con*	*avgs*
300	100.0	8 965	96.7	10 838	100.0	14 501	100.0	24 702	93.3	55 159
600	96.7	32 242	100.0	39 056	100.0	52 675	96.7	92 717	90.0	22 8427
900	90.0	69 208	93.3	83 357	86.7	112 598	76.7	201 642	60.0	495 154
1200	96.7	118 483	90.0	144 259	86.7	195 396	83.3	338 903	-	823 612
1500	90.0	179 932	93.3	217 230	86.7	301 845	-	526 029	-	1 352 817
1800	83.3	252 972	93.3	308 373	-	423 450	-	716 598	-	1 818 608
2100	76.7	337 443	-	409 771	-	563 840	-	969 069	-	2 431 715

These two sets are implemented as height-balanced binary search trees in order to allow logarithmic-time operations. If the implementation does not terminate before one hour of CPU-time, or if RAM resources available are exceeded, the run is cancelled and the output is omitted. The code is written in C++.

BUOP instances are defined with two parameters: problem size (n) and correlation between profit and weight vectors (ρ). Both parameters affect the size of the efficient set. The positive (resp. negative) data correlation will increase (resp. decrease) the degree of conflict between the two criteria. The size of the instances ranges from 300 to 3000 and the correlations are $\rho \in \{-0.8, -0.4, 0.0, 0.4, 0.8\}$. Profit and weight integer values are generated randomly according to a uniform distribution in $[1, M/n]$, where M denotes the maximum possible integer value. The generation of correlated data follows the procedure given by Verel et al. [12]. For each problem size and each correlation degree, 30 different and independent instances are randomly generated.

Two neighborhood structures are considered for the experimental analysis: the 1-*flip* and 1-*flip*-*exchange* neighborhoods. Two feasible solutions are 1-*flip* neighbors if they differ exactly on one assignment. In other words, a given neighbor can be reached by adding or removing one item from a given solution. Hence, this neighborhood structure is directly related to the Hamming distance between binary strings. The 1-*flip*-*exchange* neighborhood is an extension of the neighborhood 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 size of the neighborhood is linear with n for the 1-*flip* neighborhood structure, while it is quadratic in the case of 1-*flip*-*exchange*. Both neighborhoods coincide with the neighborhood structures used by Gorski et al. [5] for a similar class of problems.

For the connectedness analysis, MDP-BUOP outputs the efficient set for every instance. For each neighborhood structure, an adjacency matrix is built, indicating whether each two efficient solutions are neighbors or not. Based on this matrix, the connectedness of the corresponding graph is tested. Since this

Algorithm 1 Pareto Local Search

Input: $n \in \mathbb{N}, p, w \in \mathbb{N}^n, x_0 := \mathbf{0}$.
Output: Set V_T

```
1: V_F := {x_0}
2: V_T := ∅
3: while V_F ≠ ∅ do
4:    select x* from V_F
5:    V_F := V_F \ {x*}
6:    V_T := V_T ∪ {x*}
7:    for all x ∈ N(x*) do
8:       if {x' | x' ∈ V_F ∪ V_T, (p(x'), −w(x')) ≥ (p(x), −w(x))} = ∅ then
9:          V_F := {x' | x' ∈ V_F, (p(x), −w(x)) ≱ (p(x'), −w(x'))}
10:         V_T := {x' | x' ∈ V_T, (p(x), −w(x)) ≱ (p(x'), −w(x'))}
11:         V_F := V_F ∪ {x}
12: return V_T
```

analysis involves a large usage of memory resources (more than 2Gb for large-size instances), only results for a limited number of instance sizes and correlation values are presented. Table 1 gives the percentage of instances with the efficient set that is connected with respect to the 1-*flip* neighborhood, as well as the average size of the efficient set, rounded to the nearest integer. Although the correlation in the input data influences the size of the efficient set, it does not seem to affect the connectedness results. However, the proportion of instances with a connected efficient set slightly decreases with the increase of the instance size. Finally, for those set of instances, the efficient set is always connected with respect to the 1-*flip*-*exchange* neighborhood.

2.3 Local Search for BUOP

The local search algorithm for BUOP (PLS-BUOP) is based on the Pareto Local Search [9]. The pseudo-code is given in Algorithm 1. For simplification purpose, it is assumed that all feasible solutions have a distinct image in the criterion 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-BUOP starts with an efficient 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 neighborhood 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 it by starting from at least one efficient solution (with a proper change of the conditions in Algorithm 1) [9]. In the following, some particular details of the implementation are described.

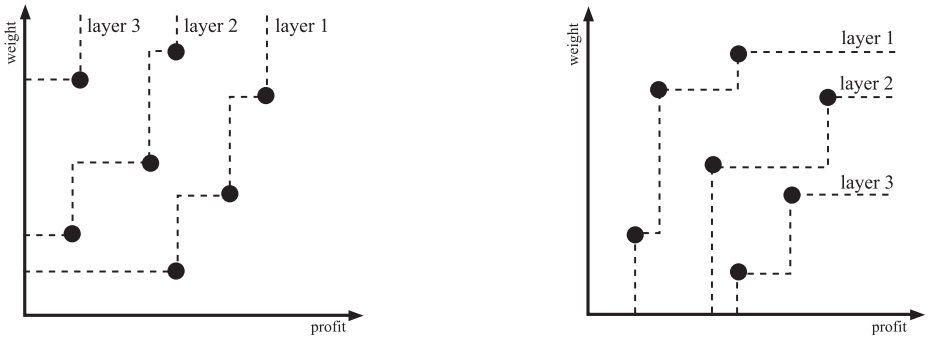

Fig. 1. Initial dominance-depth ranking of items for the exploration of the neighborhood (left: ranking for adding, right: ranking for deleting).

- *Initialization.* The algorithm starts with the efficient solution $x_0 = \mathbf{0}$. This solution maps, in the criteria space, to an extreme point from the nondominated set, with null profit and weight values.
- *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 ranks the items into different layers, with respect to the *dominance-depth ranking* [4]. Two different ranks are required to cover the cases of adding and deleting an item. This step is illustrated in Fig. 1. Let L_i denote the set of items in the i-th layer and let $u \in L_i$ and $v \in L_j$, $j > i$. Then, for the case of the addition, it holds that $(p_v, -w_v) \not\geq (p_u, -w_u)$. Therefore, the neighborhood exploration starts by examining the items in the first layer, and proceeds with the items in the subsequent layers. Within the same layer, the exploration follows the nondecreasing order of the weights. The exploration stops after verifying that no item of a given layer belongs to the current solution. Indeed, any neighboring solution constructed from the subsequent layers is dominated by at least one neighboring solution built from this layer. For the case of deletion, a similar reasoning applies with the corresponding changes.
- 1-*flip-exchange neighborhood exploration.* For this neighborhood, the same reasoning as above applies when adding or removing an item. 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 tuple records the profit and weight difference with respect to each different item j. Then, the $n - 1$ tuples are sorted in terms of dominance-depth ranking. When considering the exchange of item i with another item, the exploration follows the order given by the dominance-depth ranking, with the nondecreasing order of the weighs for tuples within the same layer; the exploration stops once the items corresponding to all tuples of a given layer can be exchanged with i. The exploration is iterated for every item in the knapsack.

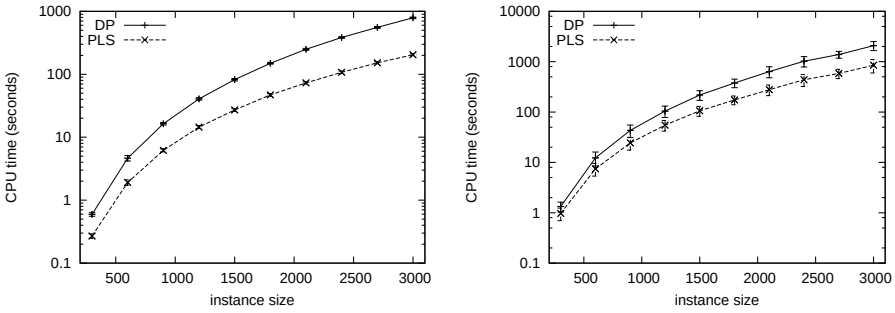

Fig. 2. CPU time in seconds (average value and standard deviation, given in log scale) of MDP-BUOP and PLS-BUOP using the 1-*flip* neighborhood for instances with data correlation $\rho = -0.4$ (left) and $\rho = 0.4$ (right).

– *Data structures.* Archives V_F and V_T are implemented as height-balanced binary search trees. The removal of dominated solutions follows the algorithm of Kung et al. [7].

MDP-BUOP and two PLS-BUOP versions, each one for each neighborhood structure, were run on the instances described in Section 2.2. The implementation of MDP-BUOP was modified in order to output a minimal complete set. All the algorithms share the same programming language, data structures, compilation options and were run in the same machine with a time bound of one hour. The PC used for the experiments was an iMac with Mac OS X v10.5.5 (Leopard), 2.4 GHz Intel Core with 4MB L2 Cache and 2GB SDRAM. All codes were compiled with g++ version 4.0.1 using the -O3 flag. The CPU-time taken by MDP-BUOP and PLS-BUOP with 1-*flip* neighborhood for instances with correlation $\rho = -0.4$ and $\rho = 0.4$ is reported in Fig. 2. In order to distinguish the differences of performance, the CPU-time is presented in log scale. Similar relationship between performance of both implementations was obtained for the remaining correlation values. Table 2 presents the percentage of efficient solutions and the total number of solutions returned by PLS-BUOP using 1-*flip* neighborhood, averaged over results obtained in 30 instances for each size and correlation. Since PLS-BUOP using the 1-*flip*-*exchange* neighborhood took always more time than MDP-BUOP, the results of the former are omitted.

Although the experimental analysis indicated the existence of instances with unconnected efficient set with respect to the 1-*flip* neighborhood (see Table 1), 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. 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-BUOP and MDP-BUOP in

Table 2. Average percentage of efficient solutions and average number of solutions found by PLS-BUOP using the 1-*flip* neighborhood ($\%ef$ and $avgs$, respectively).

	$\rho = -0.8$		$\rho = -0.4$		$\rho = 0.0$		$\rho = 0.4$		$\rho = 0.8$	
size	$\%ef$	$avgs$	$\%ef$	$avgs$	$\%ef$	$avgs$	$\%ef$	$avgs$	$\%ef$	$avgs$
300	100.0	8 965	99.9	10 838	100.0	14 501.2	100.0	24 702	99.9	55 159
600	99.9	32 242	100.0	39 056	100.0	52 675.4	100.0	92 717	100.0	228 427
900	99.9	69 208	99.9	83 357	99.9	112 597.3	99.9	201 642	99.9	495 153
1200	100.0	118 483	99.9	144 259	99.9	195 395.6	100.0	338 903	99.9	823 611
1500	99.9	179 932	100.0	217 230	100.0	301 844.6	99.9	526 029	99.9	1 352 816
1800	99.9	252 972	100.0	308 373	99.9	423 450.0	99.9	716 598	99.9	1 818 607
2100	99.9	337 443	99.9	409 770	99.9	563 839.5	99.9	969 068	99.9	2 431 713
2400	100.0	434 333	99.9	530 843	100.0	719 388.3	99.9	1 300 909	-	3 358 473
2700	99.9	541 046	100.0	661 676	100.0	900 284.3	100.0	1 560 123	-	4 158 499
3000	99.9	658 680	100.0	806 236	100.0	1 100 882.2	99.9	2 022 463	-	4 846 109

terms of CPU time. However, PLS-BUOP appears to be much more efficient for negatively correlated data, while MDP-BUOP was not able to solve all the instances for positively correlated profit and weight values.

3 The Bicriteria Knapsack Problem with Bounded Cardinality

This section reports a similar analysis on a variant of the previous problem, where an additional cardinality constraint is considered.

3.1 Problem Definition

The bicriteria knapsack problem with bounded cardinality is a variant of the BUOP that is obtained by limiting the number of chosen items by a cardinality bound (k). The formulation of BKP-BC is as follows:

$$\begin{aligned} &\max\ (p(x), -w(x)) \\ &\text{s.t.} \sum_{i=1}^{n} x_i \leq k \end{aligned} \tag{3}$$

In the problem above, the operator "max" follows the same meaning as in Problem (2). The same terminology and notation will be used for this problem.

3.2 Connectedness Analysis

The efficient set is computed by means of a multicriteria dynamic programming (MDP-BKP-BC) algorithm that extends the MDP-BUOP algorithm given in Section 2.2. This algorithm has $k \times n$ 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, $i = 1, \ldots, n$, with cardinality j, $j = 1, \ldots, k$. A state $t = (t^p, t^w) \in T_{(i,j)}$ represents a feasible solution of profits 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 $i = 1, \ldots, n$ and $j = 1, \ldots, k$, with the basis cases $T_{(i,0)} := (0, 0)$, for $i = 0, \ldots, n$ and $T_{(0,j)} := (0, 0)$ for $j = 0, \ldots, k$. 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 2.2 for the MDP-BUOP algorithm. However, the implementation of MDP-BKP-BC has to keep $2(k+1)$ sets during the overall process, since at each state i, the sets $T_{(i,j)}$ and $T_{(i-1,j)}$, for $j = 0, \ldots, k$ are required. For this reason, MDP-BKP-BC should take more time than MDP-BUOP for the same instance size.

A similar set of instances to those used for the previous problem is generated. In addition to the instance size (n) and to the data correlation (ρ), several different values for the cardinality bound (k) are considered: $n/10$, $n/5$, and $n/2$. For each problem size, correlation and cardinality bound, 30 different and independent instances are generated randomly, as explained in Section 2.2. The two neighborhood structures described in Section 2.2 were also considered for this problem. The use of the 1-*flip*-*exchange* neighborhood for this problem is motivated by the conjecture that many efficient solutions may have the same cardinality, due to the additional cardinality constraint.

Due to limited memory resources, results were only obtained for instances of limited size (these instances can be inferred from Table 3). Differently from the connectedness results obtained in the first problem (see Table 1), no instance with an efficient set that is connected with respect to the 1-*flip* neighborhood was found. However, the efficient set for all the instances were connected with respect to the 1-*flip*-*exchange* neighborhood. These results corroborate those of Gorski et al. [5] for much smaller instances (up to 100 items).

3.3 Local Search for BKP-BC

Given the positive results reported in the previous section, a local search (PLS-BKP-BC) was developed under the same reasoning of PLS-BUOP (see Section 2.3). 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.

MDP-BKP-BC and two versions of PLS-BKP-BC were run in the same instances defined in the previous section. The experiments were performed in a computer cluster with 6 nodes, each with an AMD Phenom II X6 processor with 3.2GHz, 3 and 6 MB L2 and L3 Cache, respectively, and 12 MB DDR3 SDRAM. The operating system was Ubuntu 8.04 LTS. Both codes were compiled with gcc version 4.2.4 using the -O3 flag.

The CPU-time taken by the three approaches is plotted in Fig. 3 for four different instance parameter settings. In order to distinguish the differences of

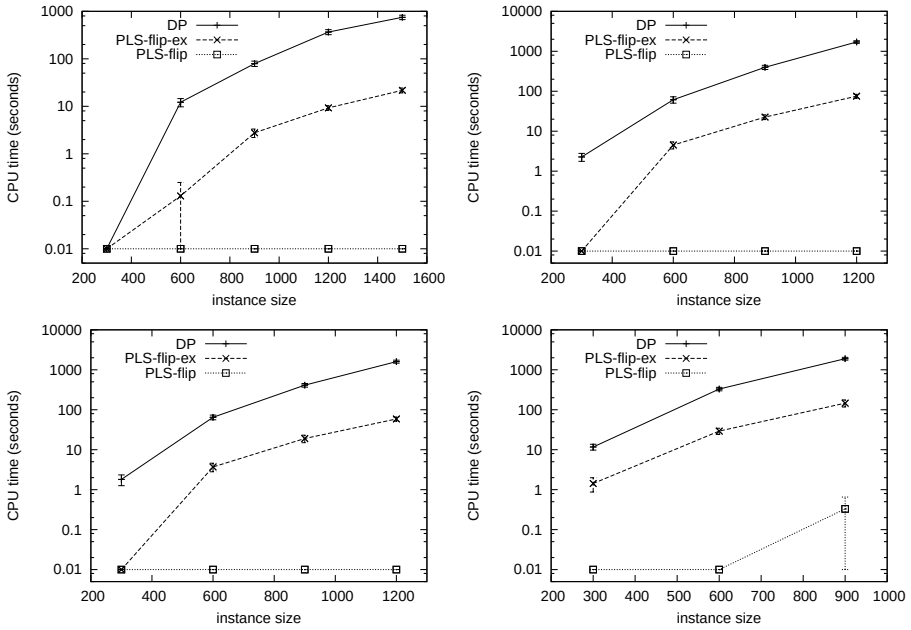

Fig. 3. CPU time in seconds (average value and standard deviation, given in log scale) of the three approaches for instances with parameters $k = n/10$ (top) and $k = n/5$ (bottom), $\rho = -0.4$ (left) and $\rho = 0.4$ (right).

performance, the CPU-time is presented in log scale. Clearly, both PLS-BKP-BC versions take much less time than MDP-BKP-BC to terminate. Similar results hold for the remaining instances. Table 3 reports the percentage of efficient solutions found by the two versions of PLS-BKP-BC. As expected from the connectedness analysis reported in the previous section, PLS-BKP-BC using the 1-*flip*-*exchange* neighborhood was always able to find a minimal complete set for all instances up to 1800 items in less than one hour of CPU-time. PLS-BKP-BC using 1-*flip* neighborhood is, in many cases, able to find more than half of a minimal complete set in less than one second.

4 Concluding Remarks

This article describes an experimental analysis on the structure of the efficient set, in terms of connectedness, for two MCO problems. Despite of the negative results reported in the literature for similar problems [3, 5], the experimental analysis for the problems investigated in this paper are quite promising. For both bicriteria versions of the unconstrained optimization problem and knapsack problem with a bounded cardinality constraint, the experiments suggest that small-sized neighborhood structures give rise to connected efficient sets quite

Table 3. Average percentage of efficient solutions found by PLS-BKP-BC using the 1-*flip* and the 1-*flip-exchange* neighborhood structures ($\%ef_1$ and $\%ef_2$, respectively).

		$\rho = -0.8$		$\rho = -0.4$		$\rho = 0.0$		$\rho = 0.4$		$\rho = 0.8$	
size	card.	$\%ef_1$	$\%ef_2$	$\%ef_1$	$\%ef_2$	$\%ef_1$	$\%ef_2$	$\%ef_1$	$\%ef_2$	$\%ef_1$	$\%ef_2$
300	$n/10$	72.6	100.0	54.1	100.0	42.0	100.0	44.2	100.0	47.7	100.0
	$n/5$	83.9	100.0	63.0	100.0	53.5	100.0	54.4	100.0	56.3	100.0
	$n/2$	95.6	100.0	88.6	100.0	85.3	100.0	83.1	100.0	84.7	100.0
600	$n/10$	75.3	100.0	50.9	100.0	41.6	100.0	41.6	100.0	47.6	100.0
	$n/5$	84.2	100.0	63.3	100.0	51.3	100.0	50.0	100.0	55.3	100.0
	$n/2$	95.4	100.0	88.3	100.0	83.1	100.0	83.4	100.0	-	-
900	$n/10$	75.6	100.0	50.1	100.0	39.6	100.0	40.5	100.0	47.0	100.0
	$n/5$	83.1	100.0	61.0	100.0	50.4	100.0	51.5	100.0	-	-
1200	$n/10$	75.6	100.0	50.5	100.0	40.2	100.0	39.9	100.0	-	-
	$n/5$	82.3	100.0	60.0	100.0	50.9	100.0	-	-	-	-
1500	$n/10$	75.4	100.0	50.5	100.0	40.3	100.0	-	-	-	-
	$n/5$	83.1	100.0	-	-	-	-	-	-	-	-
1800	$n/10$	75.4	100.0	-	-	-	-	-	-	-	-

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 the second problem. 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 approach proposed in this article is very appealing for implementation purpose, needs no definition of parameters and requires a minimum number of modifications in order to be applied to other type of knapsack problems. For instance, the same principles can be applied to the multicriteria knapsack problem (with several maximizing profit criteria and one capacity constraint) by ignoring (or penalizing) infeasible neighboring solutions. However, finding appropriate definitions of neighborhoods that give rise to a 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 MCO 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 MCO problem may provide further insights [5]. Moreover, the derivation of bounds on the run-time of multiobjective 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 [1]. This suggests that a polynomial run-time bound may be achieved by such type of algorithms.

Acknowledgements. The authors acknowledge Jochen Gorski for the discussion on the main topic of this article. 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. The third author acknowledges a CEG-IST grant from Instituto Superior Técnico (PTDC/GES/73853/2006).

References

1. Beier, R., Vöcking, B.: Probabilistic analysis of knapsack core algorithms. In: Proc. of the 15th ACM-SIAM Symposium on Discrete Algorithms (SODA 2004). pp. 468–477 (2004)
2. Ehrgott, M.: Multicriteria optimization, Lecture Notes in Economics and Mathematical Systems, vol. 491. Springer (2000)
3. Ehrgott, M., Klamroth, K.: Connectedness of efficient solutions in multiple criteria combinatorial optimization. European Journal of Operational Research 97(1), 159–166 (1997)
4. Goldberg, D.: Genetic Algorithms in Search, Optimization and Machine Learning. Addison-Wesley, Boston, MA, USA (1989)
5. Gorski, J., Klamroth, K., Ruzika, S.: Connectedness of efficient solutions in multiple objective combinatorial optimization. Tech. Rep. 102/2006, University of Kaiserslautern, Department of Mathematics (2006)
6. Gorski, J., Paquete, L.: On a particular case of the multi-criteria unconstrained optimization problem. Electronic Notes on Discrete Mathematics 36, 135–142 (2010)
7. Kung, H., Luccio, F., Preparata, F.: On finding the maxima of a set of vectors. Journal of ACM 22(4), 469–476 (1975)
8. Nemhauser, G., Ullman, Z.: Discrete dynamic programming and capital allocation. Management Science 15(9), 494–505 (1969)
9. Paquete, L., Schiavinotto, T., Stützle, T.: On local optima in multiobjective combinatorial optimization problems. Annals of Operations Research 156(1), 83–97 (2007)
10. Paquete, L., Stützle, T.: Clusters of non-dominated solutions in multiobjective combinatorial optimization: An experimental analysis. In: Multiobjective Programming and Goal Programming, Lecture Notes in Economics and Mathematical Systems, vol. 618, pp. 69–77. Springer (2009)
11. da Silva, C.G., Clímaco, J., Figueira, J.R.: Geometrical configuration of the Pareto fronteir of the bi-criteria 0-1-knapsack problem. Tech. Rep. 16/2004, INESC, Coimbra, Portugal (2004)
12. Verel, S., Liefooghe, A., Jourdan, L., Dhaenens, C.: Analyzing the effect of objective correlation on the efficient set of MNK-landscapes. In: Proc. of the 5th Conference on Learning and Intelligent OptimizatioN (LION 5). Lecture Notes in Computer Science, Springer (2011), to appear