

A Two-phase Heuristic for the Biobjective 0/1 Knapsack Problem

Luís Paquete¹, Catarina Camacho², and José Rui Figueira²

¹ Department of Computer Engineering,
CISUC – Centre for Informatics and Systems of the University of Coimbra,
University of Coimbra, Coimbra, Portugal
e-mail: `paquete@dei.uc.pt`

²CEG-IST, Center for Management Studies,
Instituto Superior Técnico,
Technical University of Lisbon, Lisbon, Portugal
`catarina.camacho@gmail.com`
`figueira@ist.utl.pt`

November 27, 2007

Abstract

We report an experimental analysis on the connectedness of the efficient set for several benchmark instances of the Biobjective 0/1 Knapsack Problem with respect to the k -Hamming neighborhood. Our results indicate that efficient solutions are strongly clustered with respect to small distance bounds. Based on these findings, we propose a two-phase heuristic that obtains a set of supported solutions in a first phase, followed by a local search procedure that collects further nondominated solutions. Our numerical results show that this approach reaches good solution quality on instances where *state-of-the-art* algorithms fail to solve.

Keywords: Multiple Objective Programming, Knapsack Problems, Local Search

1 Introduction

For a given efficient set of a instance of Multiobjective Combinatorial Optimization Problem (MCOP), a graph can be constructed such that each node represents an efficient solution

and an edge connects two nodes if the corresponding efficient solutions are neighbors for a given neighborhood [5]. We say that the efficient set is connected with respect to the neighborhood if the underlying graph is also connected, that is, there is a path between any two nodes. Worst-case results have been shown that the efficient set for many MCOPs are not connected in general with respect to different notions of neighborhood [5, 6].

We extend the notion of connectedness given above. The efficient set is represented as a complete, undirected and weighted graph G , where each node corresponds to a solution and each weight w_{ij} corresponds to the *shortest distance* between a pair of efficient solutions i and j with respect to some neighborhood. The distance between i and j is the minimum number of neighboring solutions to be visited to go from i to j . From the resulting graph, and for a given distance bound k , a disconnected graph G_k is obtained from G by extracting those edges whose weights are larger than k . A *cluster* is then a maximally connected component of G_k [10]. Note that if the number of clusters containing the efficient set is small for small distance bounds, then local search algorithms would be able to find most of the elements of the efficient set by starting from one or more efficient solutions [1, 7, 12].

In this paper we investigate the degree of *clustering* of the efficient set for the Biobjective 0/1 Knapsack Problem (BKP) with respect to the k -Hamming neighborhood; we say that two solutions are k -Hamming neighbors if their Hamming distance is less than or equal to k . Our experimental analysis on several benchmark instances suggests that the efficient set is strongly clustered for small Hamming distance bounds. Based on these findings, we present a two-phase heuristic that collects a set of supported solutions in the first phase, and it proceeds by exploring further nondominated solutions by means of local search using a k -Hamming neighborhood for small k . Furthermore, our numerical study on large benchmark instances shows that this approach reaches reasonably good solution quality on instances where *state-of-the-art* algorithms fail to solve.

2 The Biobjective 0/1 Knapsack Problem

The Biobjective 0/1 Knapsack Problem can be formulated as follows:

$$\begin{aligned}
\max f^1(x_1, x_2, \dots, x_n) &= \sum_{j=1}^n p_j^1 x_j \\
\max f^2(x_1, x_2, \dots, x_n) &= \sum_{j=1}^n p_j^2 x_j \\
\text{subject to:} & \\
&\sum_{j=1}^n w_j x_j \leq W \\
&x_j \in \{0, 1\}, \quad j = 1, 2, \dots, n
\end{aligned} \tag{1}$$

where, p_j^i is the profit or value of item j on objective i , $i = 1, 2$; x_j is the decision variable associated to item j , where $x_j = 1$ if item j is included in the knapsack and $x_j = 0$ otherwise; w_j is the weight of item j and W denotes the overall knapsack capacity. We assume that all the data are positive integers and that $w_j < W$, for $j = 1, 2, \dots, n$ with $\sum_{j=1}^n w_j > W$. Let $X = \{x = (x_1, x_2, \dots, x_n) : \sum_{j=1}^n w_j \leq W, x_j \in \{0, 1\}, j = 1, 2, \dots, n\}$ denote the set of feasible solutions, $X \subseteq \mathbb{N}^n$. The image of the feasible solutions when using the objective functions f^1 and f^2 defines the feasible region in the objective space, denoted here by $Z \subseteq \mathbb{N}^2$. We say a feasible solution $x \in X$ is efficient if there does not exist another feasible solution $x' \in X$ such that $z' = f(x') \leq z = f(x)$ and $z' \neq z$. The set of all efficient solutions will be denoted by X^E . We say that a vector $z \in Z$ is nondominated if there is some efficient solution $x \in X^E$ such that $z = f(x)$, and we denote the set of all nondominated vectors by Z^{ND} . Let $Conv(Z)$ denote the convex hull of set Z . Each nondominated vector $z \in Z^{ND}$ located in the frontier of $Conv(Z)$ is called *nondominated supported vector* or solutions; otherwise it is called *unsupported nondominated vector*.

Besides being an NP-hard problem, the BKP can also have an efficient set of size given by the binomial coefficient $\binom{n}{\lfloor n/2 \rfloor}$ [4]. Gorski et al. [6] showed that the efficient set for the BKP is not connected in general with respect to the Mixed Integer Linear Programming (MILP)-based neighborhood; two knapsack solutions x and x' are neighbors for the MILP-based neighborhood if x' can be obtained from x by replacing one item in x with one item in x' . The authors report an extensive experimental analysis on randomly generated BKP instances with bounded cardinality and 0/1-Multiple Choice Knapsack Problems. For the same notion of neighborhood, a local search algorithm was proposed for the BKP with bounded cardinality and equally-weighted items that is able to explore the connectedness property [12].

3 Cluster Analysis

The goal of the analysis that follows is to study the relation between the Hamming distance bound and the number of clusters of efficient solutions. We choose the benchmark instances available from the literature [2, 3] and that are described as follows: i) *Type A* instances with each p_j^1, p_j^2 , and w_j generated according to a uniform distribution in the range $[1, 1000]$; ii) *Type B* instances with each p_j^1, p_j^2 , and w_j generated according to a uniform distribution in the range $[111, 1000]$, $[p_j^1 - 100, p_j^2 + 100]$, and $[1, 1000]$, respectively, which induces a positive correlation between profits; iii) *Type C* instances with each p_j^1, p_j^2 , and w_j generated according to a uniform distribution in the range $[1, 1000]$, $[\max\{900 - p_j^1; 1\}, \min\{1100 - p_j^1; 1000\}]$, and $[1, 1000]$, respectively, which induces a negative correlation between profits. For all types of instances, W is half of the sum of the weights w_j .

Two dynamic programming algorithms proposed by Bazgan et al. [2] and by Captivo

et al. [3] were used for generating the efficient set. The first is an extension of a dynamic programming algorithm for the single objective case [8]. The second solves the BKP converted into a Biobjective Shortest Path Problem over an acyclic network. Since the original implementations only maintain the nondominated vectors during the run, we modified them in order to output also the efficient set. However, we noticed that these modifications imply much more use of memory. After some preliminaries experiments on a Intel Dual Core Pentium processor with 3 GHz and 1MB of cache and 1 GB RAM, we were only able to solve the following instance types and sizes:

- Type A, $n = \{70, 150\}$ [3];
- Type A, $n = \{100, 200\}$ [2];
- Type B, $n = \{100, 200, 250, 300\}$, extracted from Type B instances [2];
- Type C, $n = \{30, 40, 50\}$, extracted from Type C instances [2];
- Type C, $n = 100$ [2].

The numerical experiments were performed on 10 instances of each type and size. On a first phase, we computed the Hamming distance between each pair of efficient solutions. The resulting square matrix is the adjacency matrix for the complete undirected weighted graph G and from which maximally connected components are extracted for each distance bound. We implemented an efficient disjoint-set algorithm that computes the maximally connected components incrementally, and that terminates once all efficient solutions are located in a single cluster. Finally, for each distance bound, we counted the number of clusters with more than one element, as well as the number of efficient solutions inside each cluster.

Table 1 shows the average number of clusters and percentage of efficient solutions in the clusters for several distance bounds. The numerical results clearly indicate that efficient solutions are strongly clustered for small distance bounds, maximum $k = 5$. Moreover, we did not detect any significant difference between instances types, despite the fact that the number of efficient solutions differ considerably between different instance types. These results suggest that local search algorithms may be a good approach for this problem, once one or more efficient solutions are found. In the next section we describe a two-phase heuristic that takes advantage of these findings.

4 A Two-Phase Heuristic

The two-phase heuristic for the BKP is based on the local search approaches for the Biobjective Minimum Spanning Tree Problem [1, 7]. In the first phase, a set of supported solutions

is obtained by solving several weighted sum formulations. For our particular case, we modified a dynamic programming implementation for the single objective 0/1 Knapsack Problem from Pisinger [11] and implemented a dichotomic search that is able to obtain all supported extreme solutions, as described by Hamacher and Ruhe [7].

The output of the first phase is used as input for the second phase. The initial set of solutions is then extended by an iterative procedure that explores the neighborhood of its elements [9]. At each iteration, the algorithm works as follows: i) a non-visited solution s is chosen from the set of solutions S ; ii) the set of neighbors of s that are nondominated with respect to all solutions in S are added to S and those that are dominated are removed; iii) the solution s is flagged as visited. We used an AVL tree for maintaining S , which allows $O(|S| \log |S|)$ -time complexity for every update. Note that this local search always terminates [9]. In addition, if the efficient set is connected with respect to some definition of neighborhood, this local search will terminate when the efficient set is found. However, the heuristic may take exponential time to terminate if the efficient set is of exponential size.

5 Numerical Experiments

The goal of this experimental analysis is to analyse the performance of the two-phase heuristic on a benchmark set of instances. In addition, we analyse several neighborhoods for the second phase and relate their performance with different type of instances.

5.1 Neighborhood Definitions

The k -Hamming neighborhood was considered for the second phase, where two solutions are k -Hamming neighbors if their Hamming distance is less or equal than k . Some preliminary experiments indicated that searching for all neighbors for $k = 5$ was too time consuming. Therefore, we considered the k -Hamming neighborhood for $k = \{2, 3, 4\}$; we denote them by **2-opt**, **3-opt**, and **4-opt**, respectively. Note that the time complexity for finding the best neighbor is $O(n^k)$. In addition, we considered the MILP-based definition of neighborhood that we call **2h-opt**, that is, all solutions that can be obtained by exchanging x_i with x_j , $x_i \neq x_j$, $i < j$. Finally, we also considered two extensions of the **2h-opt** neighborhood: i) **3h-opt** that consists of all neighbors that can be obtained by changing the value of x_l at each **2h-opt** neighbor, $j < l$, *plus* all **2h-opt** neighbors; ii) **4h-opt** that consists of all neighbors that are obtained by changing the value of x_m at each **3h-opt** neighbor, $l < m$, *plus* all **2h-opt** and **3h-opt** neighbors. The time complexity for finding the best neighbor is $O(n^2)$, $O(n^3)$, and $O(n^4)$ for the **2h-opt**, **3h-opt** and **4h-opt** neighborhoods, respectively.

5.2 Experimental Results and Discussion

The code was written in C and compiled using GCC 4.1.2 with optimization level 3. The program was run under Debian GNU/Linux on a computer with 1GB RAM and with an Intel Dual Core Pentium processor with 3 GHz and 1MB of cache. We define a time limit of 7200 secs.; if the algorithm does not terminate within that time limit, it outputs the set of solutions obtained at that limit. For each output, we computed the following indicators: i) Number of runs that terminated before the time limit; ii) CPU-time taken by the first phase and by both phases; iii) Maximum size of the output at the end of the first phase and at the end of the second phase; iv) Percentage of efficient solutions returned at the end of the first phase and at the end of the second phase; v) ϵ -indicator [13] at the end of the second phase. We used the nondominated vectors returned by the dynamic programming approach [2] to compute the values of the ϵ -indicator.

The experimental results are shown in Table 3. We can observe that the first phase takes less than one second to terminate. However, this is not surprising as the percentage of efficient solutions that are supported is considerably low, in many cases less than 10% of the efficient set. Moreover, this percentage reduces as the instance size increases. Therefore, approaches that rely only on solving weighted scalarizations should have a very limited performance. On the other hand, the table indicates that the second phase is able to attain many unsupported solutions, more noteworthy on instances of type A and C.

The table also shows that this heuristic is able to attain around 99% of the efficient set for the smallest instances of Type A and B under the **4h-opt** neighborhood, and between 83% and 94% under the **3-opt** and **3h-opt** neighborhoods. For larger instances of Type A and C, this heuristic under the **3h-opt** neighborhood always terminates before reaching the time limit, as opposed to larger neighborhoods. Moreover, the solution quality obtained under the **3h-opt** neighborhood didn't differ too much from the solution quality obtained under the **3-opt** neighborhood. For larger instances of Type B, both **2-opt** and **2h-opt** neighborhoods are the only feasible choices. They perform extremely fast although the solution quality is relatively low as compared to the remaining larger neighborhoods. Note that results obtained with the **2h-opt** neighborhood match the negative results on the connectedness of the efficient set BKP with respect to the MILP-based neighborhood [6]. In fact, we found not even a single instance where a connected efficient set was found under the **2h-opt** neighborhood.

Finally, Table 2 shows that our two-phase heuristic needs much less memory than the dynamic programming algorithm [2], as it maintains much less solutions during the run. Using the two-phase heuristic under **3-opt** and **3h-opt** neighborhood, the maximum values for the size of the efficient set are almost reached, especially for the Type A and C instances (less than 20% deviation from the maximum values for the size of the efficient set). We remind that the dynamic programming algorithm is not able to solve instances larger than

100 items with the modification described on Section 3.

6 Concluding Remarks

The heuristic approach proposed here presents a quite reasonable performance for a wide range of instances of the Biobjective 0/1 Knapsack Problem. We remark that *state-of-the-art* exact algorithms are not able to go beyond 100 items due to memory reasons, whereas some variants of the two-phase heuristic are able to tackle problems with 4000 items. Although not tested here, the same approach can deal with more objectives. In that case, only the dichotomic search procedure has to be modified.

Acknowledgements The authors also acknowledge A. Hugot and J. L. Santos for kindly providing us the code.

References

- [1] K. Andersen, K. Jörnsten, and M. Lind. On bicriterion minimal spanning trees: An approximation. *Computers & Operations Research*, 23(12):1171–1182, 1996.
- [2] C. Bazgan, H. Hugot, and D. Vanderpooten. An efficient implementation for the 0-1 multi-objective knapsack problem. In *Proceedings of the 6th International Workshop on Experimental Algorithms (WEA 2007)*, volume 4525 of *LNCIS*, pages 406–419. Springer Verlag, 2007.
- [3] M. E. Captivo, J. Clímaco, J. R. Figueira, E. Q. V. Martins, and J. L. Santos. Solving bicriteria 0-1 knapsack problems using a labeling algorithm. *Computers & Operations Research*, 30(12):1865–1886, 2003.
- [4] C. Gomes da Silva, J. Clímaco, and J. Figueira. Geometrical configuration of the Pareto frontier of the bi-criteria 0-1-knapsack problem. Technical Report 16/2004, INESC, Coimbra, Portugal, 2004.
- [5] M. Ehrgott and K. Klamroth. Connectedness of efficient solutions in multi criteria combinatorial optimization. *European Journal of Operational Research*, 97:159–166, 1997.
- [6] J. Gorski, K. Klamroth, and S. Ruzika. Connectedness of efficient solutions in multiple objective combinatorial optimization. Technical Report 310, Friedrich-Alexander-Universität Erlangen-Nürnberg, Institut für Angewandte Mathematik, 2006.

- [7] H. V. Hamacher and G. Ruhe. On spanning tree problems with multiple objectives. *Annals of Operations Research*, (52):209–230, 1994.
- [8] G. L. Nemhauser and Z. Ullman. Discrete dynamic programming and capital allocation. *Management Science*, 15(9):494–505, 1969.
- [9] L. Paquete, T. Schiavinotto, and T. Stützle. On local optima in multiobjective combinatorial optimization problems. *Annals of Operations Research*, pages 83–97, 2007.
- [10] L. Paquete and T. Stützle. Clusters of non-dominated solutions in multiobjective combinatorial optimization. *Lecture Notes in Economics and Mathematical Systems*. Springer Verlag, 2008. In press.
- [11] D. Pisinger. A minimal algorithm for the 0-1 knapsack problem. *Operations Research*, 45:758–767, 1997.
- [12] Walker and O’Sullivan. Connecting efficient knapsacks - Experiments with the equally-weighted bi-criteria knapsack problem. In *ORSNZ Conference*, Auckland, New Zealand, 2004.
- [13] E. Zitzler, L. Thiele, M. Laumanns, C. M. Fonseca, and V. Grunert da Fonseca. Performance assessment of multiobjective optimizers: An analysis and review. *IEEE Transactions on Evolutionary Computation*, 7(2):117–132, 2003.

Type	Size	$k = 2$		$k = 3$		$k = 4$		$k = 5$	
		$\#G_k$	$\%opt$	$\#G_k$	$\%opt$	$\#G_k$	$\%opt$	$\#G_k$	$\%opt$
A	70	10.7	71%	2.0	96%	1.1	100%	1.0	100%
	100	25.4	75%	3.4	95%	1.0	100%	1.0	100%
	150	41.3	67%	6.2	94%	1.2	100%	1.0	100%
	200	89.0	64%	14.5	92%	1.3	100%	1.1	100%
B	100	1.1	58%	1.2	86%	1.0	98%	1.0	100%
	200	2.0	50%	1.9	74%	1.3	97%	1.0	100%
	250	2.8	39%	2.9	68%	1.2	95%	1.0	100%
	300	4.2	38%	5.1	75%	1.4	99%	1.0	100%
C	30	4.9	94%	1.4	99%	1.2	100%	1.0	100%
	40	7.5	93%	2.2	99%	1.2	100%	1.0	100%
	50	11.4	92%	1.8	98%	1.1	100%	1.0	100%
	100	52.3	87%	15.5	95%	2.4	100%	1.0	100%

Table 1: Average number of clusters ($\#G_k$) and percentage of efficient solutions in clusters ($\%opt$) for several distance bounds.

Type	Size	$ X^{ND} $	[2]	2h-opt	2-opt	3h-opt	3-opt	4h-opt
A	100	251	17134.7	205	206	232	243	250
	300	1651	898524.7	1073	1073	1538	1587	1645
	500	2997	5120514.7	2022	2022	2706	2810	371
	700	5939	18959181.7	3393	3393	5033	5003	213
B	1000	218	134107.2	88	88	147	165	50
	2000	630	1595436.1	239	239	380	380	45
	3000	1140	6578947.2	456	456	456	456	63
	4000	1752	18642759.0	722	722	722	722	87
C	100	737	103921.5	661	663	726	726	736
	300	3297	3481238.4	2654	2654	2855	2902	1925
	500	9029	21282280.5	6852	6852	7375	7375	345

Table 2: Maximum values for the size of the efficient set ($|X^{ND}|$), average values for the maximum number of solutions maintained during the run of the algorithm from Bazgan et al. [2], and maximum values for the maximum number of solutions maintained during the run of the two-phase heuristic, for several neighborhoods. The values are computed over each group of 10 instances of a given type and size.

	A				B				C			
	100	300	500	700	1000	2000	3000	4000	100	300	500	
FP	Time	0.00	0.01	0.03	0.06	0.04	0.12	0.20	0.31	0.00	0.02	0.06
	% Z ND	12.6	4.9	3.4	2.4	13.8	8.1	5.8	4.9	5.0	2.3	1.6
2h-opt	Time	0.01	0.94	6.23	23.50	0.74	9.03	44.90	134.44	0.07	3.36	27.12
	%runs	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
	% Z ND	56.6	31.6	22.5	16.6	23.0	12.9	8.8	7.7	75.0	49.2	36.1
	I _ε	1.00231	1.00050	1.00028	1.00017	1.00006	1.00003	1.00002	1.00001	1.00381	1.00185	1.00113
2-opt	Time	0.02	1.32	8.45	31.10	0.99	11.89	58.39	174.43	0.10	4.74	36.72
	runs	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
	% Z ND	57.4	31.7	22.6	16.7	23.0	12.9	8.8	7.7	75.1	49.2	36.1
	I _ε	1.00218	1.00050	1.00028	1.00018	1.00006	1.00003	1.00002	1.00001	1.00358	1.00172	1.00112
3h-opt	Time	0.32	69.08	749.98	4063.90	238.45	5872.18	-	-	2.03	312.54	4079.12
	%runs	100.0	100.0	100.0	100.0	100.0	70.0	0.0	0.0	100.0	100.0	100.0
	% Z ND	83.0	78.8	73.0	69.3	50.9	39.0	9.2	5.6	90.5	69.2	54.2
	I _ε	1.00119	1.00028	1.00014	1.00009	1.00003	1.00002	1.00007	1.00006	1.00122	1.00085	1.00032
3-opt	Time	0.71	143.07	1546.87	6607.93	588.31	-	-	-	4.00	623.15	6184.14
	%runs	100.0	100.0	100.0	20.0	100.0	0.0	0.0	0.0	100.0	100.0	30.0
	% Z ND	94.2	85.7	83.0	70.6	62.5	27.6	7.3	5.4	92.7	72.9	51.7
	I _ε	1.00084	1.00026	1.00012	1.00009	1.00003	1.00007	1.00007	1.00006	1.00124	1.00080	1.00181
4h-opt	Time	6.58	3846.56	-	-	-	-	-	-	29.98	-	-
	%runs	100.0	100.0	0.0	0.0	0.0	0.0	0.0	0.0	100.0	0.0	0.0
	% Z ND	99.0	99.2	12.6	3.6	22.8	8.2	5.6	4.9	99.7	60.7	3.9
	I _ε	1.00032	1.00001	1.00353	1.00261	1.00013	1.00008	1.00007	1.00006	1.00050	1.00921	1.00765

Table 3: Average CPU-time (Time) and average percentage of nondominated vectors found ($\%|Z^{ND}|$) for the first phase (FP); average CPU-time, average number of runs that terminated within the time limit ($\%runs$), average percentage of nondominated vectors found, and average values for the ϵ -indicator (I_ϵ), for each neighborhood. The average CPU-time is censored whenever exist runs that exceed the time limit.