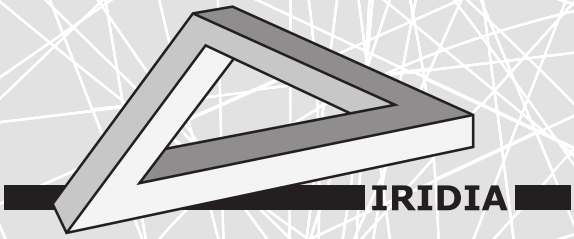

Université Libre de Bruxelles

Institut de Recherches Interdisciplinaires
et de Développements en Intelligence Artificielle

An Analysis of Local Search for the Bi-objective Bidimensional Knapsack Problem

L. C. T. Bezerra, M. López-Ibáñez, and T. Stützle

IRIDIA – Technical Report Series

Technical Report No.
TR/IRIDIA/2013-005

March 2013
Last revision: April 2013

IRIDIA – Technical Report Series
ISSN 1781-3794

Published by:

IRIDIA, *Institut de Recherches Interdisciplinaires et de Développements en Intelligence Artificielle*
Université Libre de Bruxelles
Av F. D. Roosevelt 50, CP 194/6
1050 Bruxelles, Belgium

Technical report number TR/IRIDIA/2013-005

Revision history:

TR/IRIDIA/2013-005.001 March 2013
TR/IRIDIA/2013-005.002 April 2013

An Analysis of Local Search for the Bi-objective Bidimensional Knapsack Problem

Leonardo C. T. Bezerra, Manuel López-Ibáñez, and Thomas Stützle

IRIDIA, Université Libre de Bruxelles, Brussels, Belgium
{lteonaci,manuel.lopez-ibanez,stuetzle}@ulb.ac.be

Abstract. Local search techniques are increasingly often used in multi-objective combinatorial optimization due to their ability to improve the performance of metaheuristics. The efficiency of multi-objective local search techniques heavily depends on factors such as (i) neighborhood operators, (ii) pivoting rules and (iii) bias towards good regions of the objective space. In this work, we conduct an extensive experimental campaign to analyze such factors in a Pareto local search (PLS) algorithm for the bi-objective bidimensional knapsack problem (bBKP). In the first set of experiments, we investigate PLS as a stand-alone algorithm, starting from random and greedy solutions. In the second set, we analyze PLS as a post-optimization procedure.

1 Introduction

The efficiency of many successful heuristic algorithms for combinatorial optimization problems (COPs) is based on the proper use of local search procedures. In fact, many metaheuristics have incorporated the possibility of using local search for example as daemon actions in ant colony optimization (ACO) and as improvement procedures in genetic algorithms.

Pareto local search (PLS) [13] is a straightforward but effective extension of single-objective local search to multi-objective problems. Given a set of solutions, a PLS algorithm consists of selecting one solution at a time and exploring its neighborhood, thus, generating new solutions. These new solutions are added to the initial set, dominated solutions are eliminated, and the algorithm continues until each of the solutions in the solution set has been explored.

The performance of PLS algorithms usually tends to depend on (i) the quality of the input solutions, (ii) the definition of the neighborhood structure, (iii) the pivoting rule used for exploring of the neighborhood and the possible use of candidate lists, and (iv) restrictions on the set of solutions to keep the runtimes manageable. For such reasons, PLS algorithms are well suited for analyzing the impact of design features on the performance of local search procedures for multi-objective combinatorial optimization problems (MCOPs).

In this paper, we implement a PLS algorithm for the bi-objective bidimensional knapsack problem (bBKP), using the local search components commonly found in the literature. Full factorial designs are used to investigate factors and

their eventual interactions. In the first set of experiments, we investigate PLS as a stand-alone optimization method. The experimental setup used aims at isolating the effect of the initial solution set, and the effect of the neighborhood size. Results confirm PLS's dependence on high-quality input solutions, and that large neighborhoods have to be combined with candidate lists to limit exploration and keep runtimes reasonable. In the second set of experiments, we analyze PLS as a post-optimization method. Two algorithms are used for generating input solutions: (i) a simply greedy procedure, and (ii) AutoMOACO, the best-performing multi-objective ant colony optimization (MOACO) algorithm for the bBKP. Results show that PLS is able to significantly improve the quality and size of the approximation fronts generated by both algorithms.

The remainder of this paper is organized as follows. Section 2 introduces some basic definitions and presents PLS. Section 3 defines the bBKP and presents the PLS algorithm implemented. Section 4 explains the experimental setup, while the experimental results are discussed in Sections 5 and 6. Finally, conclusions and possibilities for future work are discussed in Section 7.

2 Pareto local search

Pareto local search (PLS) is a stochastic local search method for tackling multi-objective problems based on a natural extension of single-objective local search approaches [13]. To fully understand PLS, some background notions on multi-objective optimization are required.

In an MCOP, solutions are compared not based on a single objective value, but on a vector of objective values. Given a maximization problem with objectives $f^i, i = 1, \ldots, k$, a solution s is said to be better (or to *dominate*) another solution s' if $\forall i,\ f^i(s) \geq f^i(s')$ and $\exists\, i,\ f^i(s) > f^i(s')$. If neither solution dominates the other, they are said to be *nondominated*. The goal of multi-objective optimizers is to find the set of nondominated solutions w.r.t. all feasible solutions, the Pareto set. Since this may prove to be computationally unfeasible, multi-objective metaheuristics have been used to find approximation sets, i.e., sets whose image in the objective space best approximates the Pareto set image.

PLS algorithms use the concept of dominance to extend single-objective local search to multi-objective problems. Starting from an initial set of solutions $\mathcal{A}_0$ (which can also be a singleton), PLS selects at each iteration an unexplored solution $s \in \mathcal{A}$, the current set of non-dominated solutions, and explores the neighborhood of s, generating new solutions. If these new solutions satisfy the algorithm's acceptance criterion, they are added to $\mathcal{A}$. Once the neighborhood of s is explored, s is marked as explored. The algorithm stops when all solutions in $\mathcal{A}$ have been explored. The three main steps of PLS as shown in Algorithm 1 can be summarized as follows:

Selecting a solution to be explored. The method NextSolution chooses the next solution to be explored. The original PLS chooses the next solution uniformly at random. More recently, other possibilities have been considered, such as selection based on the *optimistic hypervolume improvement* [6].

Algorithm 1 Pareto Local Search

Input: An initial set of nondominated solutions $\mathcal{A}_0$
1: explored(s) $\leftarrow$ FALSE $\quad \forall\, s \in \mathcal{A}_0$
2: $\mathcal{A} \leftarrow \mathcal{A}_0$
3: **repeat**
4: $\quad s \leftarrow$ NextSolution($\mathcal{A}_0$)
5: $\quad$ **for all** $s' \in$ Neighborhood(s) **do**
6: $\quad\quad$ **if** Acceptance($s, s', \mathcal{A}$) **then**
7: $\quad\quad\quad$ explored(s') $\leftarrow$ FALSE
8: $\quad\quad\quad \mathcal{A} \leftarrow$ Update($\mathcal{A}_0, s'$)
9: $\quad\quad$ **end if**
10: $\quad$ **end for**
11: $\quad$ explored(s) $\leftarrow$ TRUE
12: $\quad \mathcal{A}_0 \leftarrow \{s \in \mathcal{A} \mid$ explored(s) = FALSE $\}$
13: **until** $\mathcal{A}_0 = \emptyset$
Output: $\mathcal{A}$

Exploring the neighborhood. Given a solution s, the method Neighborhood(s) generates the set of neighbor solutions. Two pivoting rules are useful here: (i) *first*, where the neighborhood exploration stops at the first accepted neighbor, or; (ii) *full*, where the neighborhood of s is explored fully and all possible neighbors of s are examined.

Accepting new solutions. Given a solution s and a neighbor solution s', the method Acceptance($s, s', \mathcal{A}$) determines whether s' is considered an acceptable solution or not. Two common possibilities are: (i) *dominance*, where s' is only accepted if s' dominates s, or; (ii) *nondominance*, where s' is accepted in case s' is nondominated w.r.t. $\mathcal{A}$. The first criterion generates a higher pressure towards good solutions but it may lead to early stagnation. When using nondominance, the output is likely a well distributed set, but it may lead to high computation times.

Dubois-Lacoste et al. [7] present a review of PLS, highlighting its use both as a stand-alone procedure and in hybrid algorithms. Regarding stand-alone PLS, the authors identify studies focusing on time-limited experiments [10], on anytime behavior [6] and on how to restart or continue the search after PLS converges [1, 9, 4]. Regarding PLS as a post optimization procedure, Dubois-Lacoste et al. [5] have proposed algorithms that are currently state-of-the-art for several bi-objective flowshop problems.

3 Applying PLS to the bBKP

The bi-objective bidimensional knapsack problem is a widely-used bi-objective benchmark problem [14, 12], and is a special case of the general multi-objective multidimensional knapsack problem (moMKP), which is formalized as follows:

$$\max\ f^c(x) = \sum_{i=1}^{n} p_i^c x_i \quad c = 1, \ldots, k \qquad \text{s.t.} \ \sum_{i=1}^{n} w_i^j x_i \leq W_j \quad j = 1, \ldots, m \quad (1)$$

where each item i has k profits and m costs, f^c is the c-th component of the objective vector f, n is the number of items, p_i^c is the c-th profit of item i, w_i^j is

Table 1. Methods used for computing the weights used by SolutionOrdering.

Method	Formula	Description
equal	$\lambda = 0.5$	equal weights
random-discrete	$\lambda \in \{0, 1\}$	uniformly randomly chosen
random-continuous	$\lambda \in [0, 1]$	uniformly randomly chosen
largest-gap	$\lambda = i$, $min(w_s^i)$	privileges dimension with more free space
smallest-gap	$\lambda = i$, $max(w_s^i)$	privileges dimension with less free space
highest-profit	$\lambda = i$, $max(f^i(s))$	privileges objective with the highest value
lowest-profit	$\lambda = i$, $min(f^i(s))$	privileges objective with the lowest value
proportional-same	$\lambda = w_s^1/(w_s^1 + w_s^2)$	proportional to the loads
proportional-opposite	$\lambda = w_s^2/(w_s^1 + w_s^2)$	inversely proportional to the loads

the j-th cost of item i, W_j is the j-th capacity of the knapsack, and $x_i \in \{0, 1\}$ defines if item i is included in the knapsack ($x_i = 1$) or not ($x_i = 0$). The set of feasible solutions is $X \subseteq \{0, 1\}^n$. The bBKP is a special case of the moMKP where $k = m = 2$.

In this paper, a solution x for the bBKP is also represented as a list s of size n, where the first $n_s \leq n$ items are considered to be in the knapsack. Furthermore, each solution has as an associated profit vector $\boldsymbol{p}_s = (p_s^1, p_s^2)$, where $p_s^c = \sum_{i=1}^{n} p_i^c x_i^s$, $c = 1, 2$, and a load vector $\boldsymbol{w}_s = (w_s^1, w_s^2)$, where $w_s^j = \sum_{i=1}^{n} w_i^j x_i^s$, $j = 1, 2$.

Two methods have been implemented for generating the initial solution(s) for PLS: (i) *random*, where one or more random solutions are generated, and (ii) *greedy*, where a set of greedy solutions is generated. Random solutions are generated by choosing an item uniformly at random at each construction step, until no more items can be added due to the capacity constraints. When using greedy solutions, a set of linearly uniformly distributed weights $\Lambda = \{\lambda_1, \ldots, \lambda_z\}$ is generated, where z is the number of input solutions. For each $\lambda \in \Lambda$, a greedy solution is generated using one of the following heuristic functions [12]:

$$\eta_1(i) = \frac{\lambda\, p_i^1 + (1 - \lambda)\, p_i^2}{\sum_{j=1}^{m} w_i^j} \qquad \eta_2(i) = \frac{\lambda\, p_i^1 + (1 - \lambda)\, p_i^2}{\sum_{j=1}^{m} \frac{w_i^j}{W_j - w_s^j + 1}} \tag{2}$$

All actions related to neighborhood exploration are encapsulated in the procedure Neighborhood. This procedure systematically explores the neighborhood of a solution s, returning a set of neighbors. The neighborhood operator used is the r-remove operator, which removes up to r items from the knapsack. In our solution representation, removing one item at the i-th position of the list means exchanging it with the last selected item, i.e., the item at position n_s of the list, and decreasing the value of n_s by one.

Solutions are reconstructed by filling the knapsack with items found at positions $i = n_s + r, \ldots, n$ of the list, in the order they appear. Since biasing the search

is important, in the beginning of the algorithm items not selected in the input solution are ordered. Formally, given an input solution s, let $\mathcal{IN}(s) = \{i \mid s_i = 1\}$ be the set of n_s items inside the knapsack and $\mathcal{OUT}(s) = \{i \mid s_i = 0\}$ be the set of items outside the knapsack. The procedure SolutionOrdering is used to order items $x_i \in \mathcal{OUT}(s)$ in a nondecreasing order according to their heuristic value. The heuristics used for this ordering are the same presented in Eq. 2. The weights used by the heuristic functions are generated on a *per solution* basis. Given a solution s, we tested nine methods for computing λ (Table 1).

For efficiency, candidate lists are used to constrain the set of items that are considered for removal; in other words, items that are not member of the candidate list are never considered for removal in a current solution. Given a solution s, the candidate list of items for removal contains the L last items of the list, i.e., items at positions $i, n_s - L < i \leq n_s$. Two methods have been used for determining parameter L: (i) *all*, where $L = n_s$, and (ii) *input*, where L is input by the user.

The pseudocode for Neighborhood can be seen on Algorithm 2. C_r stands for a combination of r items, whereas C_* stands for a combinations of any number of items. Accepted checks if s and s' are nondominated. Finally, the search is controlled by one of the following pivoting-rules:

1. *remove-first*: given that the first accepted neighbor is generated by the removal of the item found at the i-th position of s, $n_s - L < i \leq n_s$, Neighborhood does not explore the insertion possibilities generated by the removal of items found at the j-th position of s, $\forall j, n_s - L < j < i$;
2. *remove-full*: Neighborhood explores the insertion possibilities generated by the removal of each of the items in the candidate list.
3. *insert-first*: given the insertion possibilities generated by the removal of an item x_i, Neighborhood stops at the first combination of items that produces an accepted neighbor.
4. *insert-full*: given the insertion possibilities resulting from the removal of an item x_i, Neighborhood generates all acceptable neighbors.

The pseudocode for the bBKP-PLS algorithm can be seen on Algorithm 3. The algorithm initially generates (reads) the input set of solutions (line 1). Then, a weight λ is generated for each solution $s \in \mathcal{A}_0$ (line 3), and used in the SolutionOrdering procedure (line 4). Finally, PLS is called (line 6). In this paper, we consider random selection of solutions in PLS and acceptance based on nondominance. In addition, when Neighborhood checks the nondominance between the current solution and each of its neighbors, it also checks nondominance w.r.t. to the set $\mathcal{A}$. This is done to ensure that $\mathcal{A}$ is extended by at least one nondominated solution if such a solution exists in the neighborhood of s.

4 Experimental setup

For the analysis of PLS, we use 50 bBKP instances for each size $n \in \{100, 250, 500, 750\}$ [2] as a test set. We use a full factorial design, and each configuration

Algorithm 2 Procedure Neighborhood

Input: An input solution s
$\mathcal{N} \leftarrow \emptyset$
for all $C_r \in$ candlist(s) **do**
  $s' \leftarrow$ Remove(s, C_r)
  for all $C_* \in \mathcal{OUT}(s)$ **do**
    $s'' \leftarrow$ Insert(s', C_*)
    if Accepted(s'', s') **then**
      $\mathcal{N} \leftarrow \mathcal{N} \cup s''$
      if *insertion-first* **then**
        found $\leftarrow$ true; break;
      end if
    end if
  end for
  if found and *removal-first* **then**
    break
  end if
end for
Output: A set of neighbor solutions $\mathcal{N}$

Algorithm 3 bBKP-PLS

Input: An input method input $\in \{random, greedy\}$
$\mathcal{A}_0 \leftarrow$ InitialSolutions(input)
for all $s \in \mathcal{A}_0$ **do**
  $\lambda \leftarrow$ GenerateLambda(s)
  SolutionOrdering(s, λ)
end for
$\mathcal{A} \leftarrow$PLS(A_0)
Output: $\mathcal{A}$

is run 10 times per instance. All parameters are analyzed both individually and for possible interactions through plots of the median hypervolume [14] of the approximation sets they produce.

For the computation of the hypervolume measure of an approximation set, a reference point is required. Here, we first normalize the objective vectors of all sets generated by all runs of all configurations separately for each instance. This normalization also transforms the original maximization problem into a minimization one, with objective values in the range $[1, 2]$, due to requirements of the software we use [8]. We use the point $(2.1, 2.1)$ as the reference point.

When comparing two algorithms using boxplots of their hypervolumes or plots of the differences of their empirical attainment functions (EAFs) [11], we use the four instances by Zitzler and Thiele [14] of sizes $n \in \{100, 250, 500, 750\}$, called ZTZ instances, and we run the algorithms 25 independent times per instance. When more algorithms are simultaneously compared, we use the boxplots of the hypervolumes for a sample of 4 instances of each size: the ZTZ instances plus three instances from the test set used for the analysis of PLS.

All algorithms are implemented in `C` and all experiments are run on a single core of Intel Xeon E5410 CPUs, running at 2.33GHz with 6MB of cache size under Cluster Rocks Linux version 6.0/CentOS 6.3. In the paper, only few representative results are given. The complete set of results are made available as a supplementary page [3].

Table 2. Parameter values used for the analysis of stand-alone PLS.

Parameter	Values
η	η_1, η_2
λ	*equal, random-discrete, random-continuous, largest-gap, smallest-gap, highest-profit, lowest-profit, proportional-same, proportion-opposite*
candidate list	*all*, $L = 15$, $L = 30$, $L = 50$
removal rule	*removal-first, removal-full*
insertion rule	*insertion-first, insertion-full*

5 Experiments with stand-alone PLS

In this section, we present the results of the analysis of stand-alone PLS. To properly isolate the effect of the neighborhood operator, this analysis is divided in two stages: (i) experiments removing one item ($r = 1$), and (ii) experiments removing more than one item ($r > 1$). The parameter space used for this analysis comprises all possible values previously described, summarized in Table 2.

5.1 Removing a single item

The different possibilities of choosing the weights (λ) and the heuristic information (η) do not have a strong influence on the results, hence, we do not present here a detailed analysis of these parameters. Instead, we analyze the initialization method, the length of the candidate list, and the removal and insertion pivoting rules. The results presented here are aggregated across all possible settings of λ and η. Detailed results can be found on the supplementary material.

We first analyze stand-alone PLS initialized with a single random solution. Fig. 1 (left) shows the median hypervolume on the y-axis, and the instances grouped by sizes on the x-axis. The lines represent the median hypervolume obtained by different configurations, and are ordered according to performance on the larger instance. It is clear that using candidate lists when $r = 1$ is a bad decision, since the hypervolume quickly degenerates as the instance size grows. However, as shown on Fig. 1 (right), the runtimes of configurations that do not use candidate list tend to grow very quickly (y-axis).

The analysis of PLS starting from greedy solutions requires an additional parameter, namely the number of weights used for generating input solutions. Experiments were conducted for 2, 10 and 50 input weights, and this parameter proved critical for the performance of the algorithm. When only two input weights are used, the performance of the algorithm is really poor and similar to when a random initial solution is used. On the other hand, using 50 weights not only improves the final result quality, but also helps the algorithm converge faster compared to other settings.

Therefore, we focus on the experiments using 50 initial weights for generating the greedy solutions. Also in this case, not using candidate lists leads to long runtimes. However, the solution quality does not degenerate when larger values

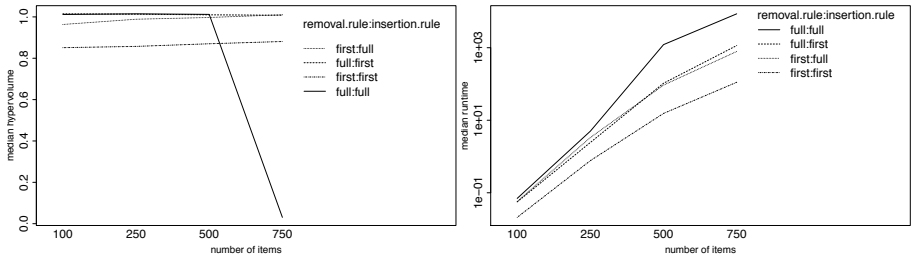

Fig. 1. Median hypervolume (left) and runtime (right) of different combinations of pivoting rules for PLS starting from a random initial solution and $r = 1$.

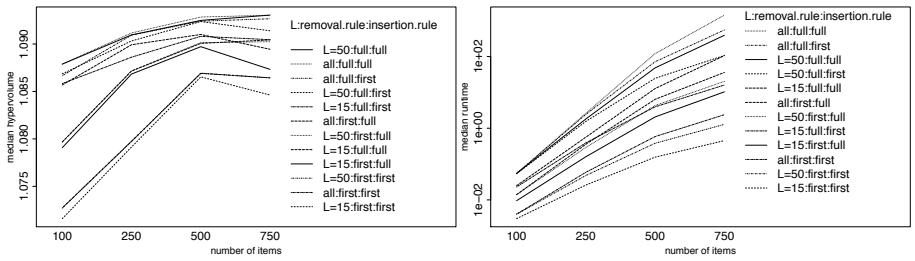

Fig. 2. Median hypervolume (left) and runtime (right) of different combinations of pivoting rule and candidate list sizes for PLS starting from greedy solutions and $r = 1$.

of L are adopted. Fig. 2 (left) shows the median hypervolume (y-axis) grouped by instance sizes (x-axis). The best performing versions are the ones that use $L = \{30, 50\}$, *removal-full* and *insertion* $\in \{$*first*, *full*$\}$ ($L = 30$ not shown here due to space reasons). When analyzing runtimes (see also Fig. 2, right), a candidate list of size 50 combined with removal-full and insertion-first is a setting that takes computation times similar to those that are used as time limits in the analysis of state-of-the-art algorithms [2, 12].

5.2 Removing more than one item

For the analysis of PLS with $r > 1$, the same parameter space used in the previous experiments is adopted. However, given that in the previous experiments with $r = 1$ we observed that the parameters used for SolutionOrdering (that is, the heuristic and the values of λ) behave very similarly, we narrowed down the number of configurations to be tested by selecting η_1 as the heuristic function and *highest-profit* as the method for defining λ. The experiments were limited to a maximum runtime of 5 hours per run.

Figure 3 shows the results for $r = 2$. In terms of solution quality, there is a big difference between configurations that use *removal-first* and *insertion-first* and the ones that do not. Moreover, the best configurations for small instance size become much worse with larger instance size. The reason is that those

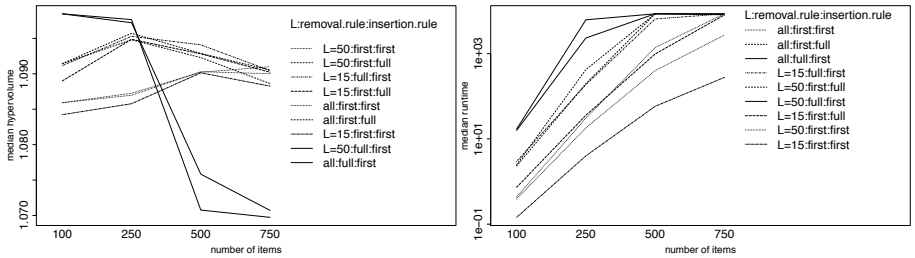

Fig. 3. Median hypervolume (left) and runtime (right) of different combinations of pivoting rule and candidate list sizes for PLS starting from greedy solutions and $r = 2$.

configurations reach the CPU-time limit of 5 hours and were stopped before completion.In fact, the only configurations with runtime lower than 1 000 seconds are the ones that either (i) use a candidate list ($L \leq 50$) combined with *removal-first* and *insertion-first*, or; (ii) use a candidate list with $L = 15$ combined with *removal-first* and *insertion-full*.

6 Experiments with PLS as post-optimization method

To analyze PLS as a post-optimization method, we start by comparing PLS against the greedy procedure used for generating its input solutions. The motivation for this comparison is to understand whether PLS is actually significantly improving the input set or simply adding more nondominated solutions. The greedy procedure is run with η_1 and $2n$ weights, which is roughly the same number of solutions expected to be found by stand-alone PLS. The parameters used by PLS are: η_1, *highest-profit,* $L = 50$*, removal-full, insertion-first*, and $r = 1$.

Fig. 4 shows that the difference between the greedy procedure and PLS (using $2n$ weights for greedy solutions) is quite strong. The approximation set identified by PLS dominates the output of the greedy procedure across the entire range of the front, which means PLS is able to substantially improve all initial solutions. In addition, PLS finds a much larger approximation set.

We also add PLS as a post-optimization procedure to AutoMOACO [2], a high-performing population-based algorithm for bBKP and run AutoMOACO using the same parameters and time limit as in the original paper. The resulting approximation set is given as input to PLS, which is then run until completion using the same parameters as above. Fig. 5 shows the EAF difference for ZTZ 750. Again, PLS is able to improve the approximation fronts over the entire objective space, while the runtimes of PLS remain low (see Table 3). Thus, PLS significantly improves the approximation obtained by AutoMOACO, incurring only a reasonable computational overhead.

To conclude this analysis, we compare all four algorithms considered in this section. Figure 6 shows the boxplot of the hypervolume for all four ZTZ instances. As expected, the greedy procedure presents the worst hypervolume val-

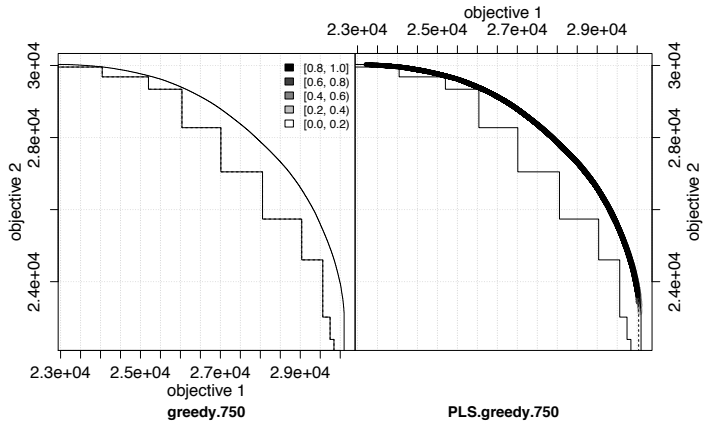

Fig. 4. EAF difference plot. Greedy solutions using $2n$ weights (Greedy) vs. PLS initialized with greedy solutions using $2n$ weights ($\text{PLS}_{\text{greedy}}$). Instance ZTZ 750.

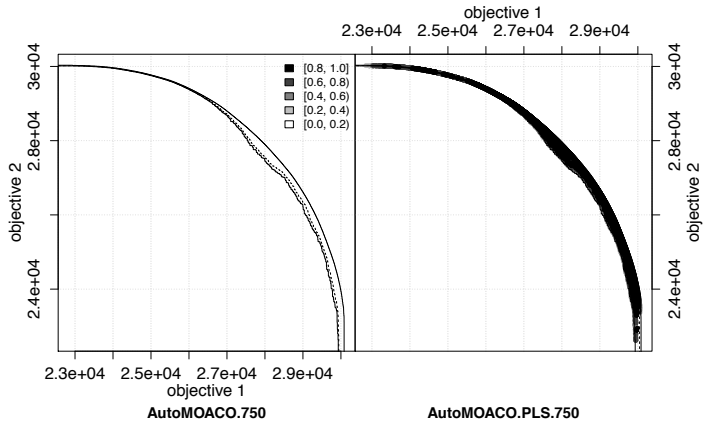

Fig. 5. EAF difference plot. AutoMOACO vs. AutoMOACO+PLS. Instance ZTZ 750.

ues. Among the remaining algorithms, AutoMOACO and stand-alone PLS perform similarly on most instances. Interestingly, although the number of solutions is greatly increased when PLS is used as post-optimization for AutoMOACO, the differences in the hypervolume are relatively small. For the largest instance, the performance of AutoMOACO decreases considerably, but PLS is able to compensate such loss.

Table 3. Average number of solutions and runtime: ZTZ instances, 25 runs.

	ZTZ 100	**ZTZ 250**	**ZTZ 500**	**ZTZ 750**
Greedy	13 (0.00s)	43 (0.01s)	69 (0.06s)	114 (0.12s)
PLS_{greedy}	98.81 (0.04s)	356.46 (1.08s)	742.38 (7.64s)	1502.2 (40.35s)
AutoMOACO	81.84 (1s)	287.84 (6.25s)	376 (26s)	273.08 (56.25s)
AutoMOACO+PLS	110.48 (1.03s)	382.76 (7.27s)	807.28 (33.48s)	1542.48 (114.6s)

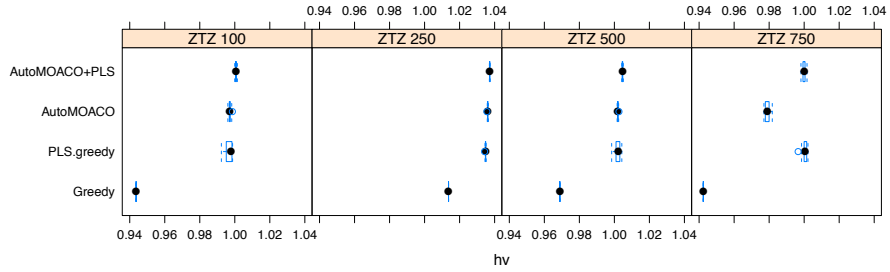

Fig. 6. Boxplot of the hypervolume indicator for ZTZ instances.

7 Conclusions and Future Work

In this paper, we have applied Pareto local search (PLS) to the biobjective bidimensional knapsack problem to analyze the impact of common local search components found in the literature, and to empirically investigate by how much it can improve over existing algorithms. Our results show that the performance of stand-alone PLS strongly depends on high-quality input solutions. However, such solutions can be generated without significant computational overhead using greedy (meta)heuristics. Large neighborhood sizes proved prohibitive w.r.t. computation time even when combined with a candidate list. Nevertheless, archiving mechanisms that constraint the size of the approximation set remain to be tested.

The insights from this research can be used to design better neighborhood operators for the bBKP. Additionally, a combination of more elaborate algorithms, such as iterated greedy with PLS could lead to state-of-the-art results.

Acknowledgments. The research leading to the results presented in this paper has received funding from the Meta-X project from the Scientific Research Directorate of the French Community of Belgium and from the FRFC project "Méthodes de recherche hybrids pour la résolution de problèmes complexes". Leonardo C. T. Bezerra, Manuel López-Ibáñez and Thomas Stützle acknowledge support from the Belgian F.R.S.-FNRS, of which they are a FRIA doctoral fellow, a postdoctoral researcher and a research associate, respectively.

References

1. Alsheddy, A., Tsang, E.: Guided Pareto local search and its application to the 0/1 multi-objective knapsack problems. In: Caserta, M., Voß, S. (eds.) MIC 2009. University of Hamburg, Hamburg, Germany (2010)
2. Bezerra, L.C.T., López-Ibáñez, M., Stützle, T.: Automatic generation of multi-objective ACO algorithms for the biobjective knapsack problem. In: Dorigo, M., et al. (eds.) ANTS 2012, LNCS, vol. 7461, pp. 345–355. Springer, Heidelberg, Germany (2012)
3. Bezerra, L.C.T., López-Ibáñez, M., Stützle, T.: An analysis of local search for the bi-objective bidimensional knapsack: Supplementary material. http://iridia.ulb.ac.be/supp/IridiaSupp2012-016/ (2012)
4. Drugan, M.M., Thierens, D.: Path-guided mutation for stochastic Pareto local search algorithms. In: Schaefer, R., et al. (eds.) PPSN XI, LNCS, vol. 6238, pp. 485–495. Springer, Heidelberg, Germany (2010)
5. Dubois-Lacoste, J., López-Ibáñez, M., Stützle, T.: A hybrid TP+PLS algorithm for bi-objective flow-shop scheduling problems. Computers & Operations Research 38(8), 1219–1236 (2011)
6. Dubois-Lacoste, J., López-Ibáñez, M., Stützle, T.: Pareto local search algorithms for anytime bi-objective optimization. In: Hao, J.K., Middendorf, M. (eds.) EvoCOP 2012, LNCS, vol. 7245, pp. 206–217. Springer, Heidelberg, Germany (2012)
7. Dubois-Lacoste, J., López-Ibáñez, M., Stützle, T.: Combining two search paradigms for multi-objective optimization: Two-Phase and Pareto local search. In: Talbi, E.G. (ed.) Hybrid Metaheuristics, Studies in Computational Intelligence, vol. 434, pp. 97–117. Springer Verlag (2013)
8. Fonseca, C.M., Paquete, L., López-Ibáñez, M.: An improved dimension-sweep algorithm for the hypervolume indicator. In: CEC 2006, pp. 1157–1163. IEEE Press, Piscataway, NJ (Jul 2006)
9. Geiger, M.J.: Decision support for multi-objective flow shop scheduling by the Pareto iterated local search methodology. Computers and Industrial Engineering 61(3), 805–812 (2011)
10. Liefooghe, A., Humeau, J., Mesmoudi, S., Jourdan, L., Talbi, E.G.: On dominance-based multiobjective local search: design, implementation and experimental analysis on scheduling and traveling salesman problems. Journal of Heuristics 18(2), 317–352 (2011)
11. López-Ibáñez, M., Paquete, L., Stützle, T.: Exploratory analysis of stochastic local search algorithms in biobjective optimization. In: Bartz-Beielstein, T., et al. (eds.) Experimental Methods for the Analysis of Optimization Algorithms, pp. 209–222. Springer, Berlin, Germany (2010)
12. Lust, T., Teghem, J.: The multiobjective multidimensional knapsack problem: a survey and a new approach. Intern. Trans. in Oper. Res. 19(4), 495–520 (2012)
13. Paquete, L., Chiarandini, M., Stützle, T.: Pareto local optimum sets in the biobjective traveling salesman problem: An experimental study. In: Gandibleux, et al. (eds.) Metaheuristics for Multiobjective Optimisation, pp. 177–200. LNEMS, Springer, Berlin, Germany (2004)
14. Zitzler, E., Thiele, L.: Multiobjective evolutionary algorithms: A comparative case study and the strength Pareto evolutionary algorithm. IEEE Transactions on Evolutionary Computation 3(4), 257–271 (1999)

An Analysis of Local Search for the Bi-objective Bidimensional Knapsack Problem

Supplementary Material

Leonardo C. T. Bezerra, Manuel López-Ibáñez, and Thomas Stützle

IRIDIA, CoDE, Universit Libre de Bruxelles, Brussels, Belgium
{lteonaci,manuel.lopez-ibanez,stuetzle}@ulb.ac.be

Abstract. Local search techniques are increasingly often used in multi-objective combinatorial optimization due to their ability to improve the performance of metaheuristics. The efficiency of multi-objective local search techniques heavily depends on factors such as (i) neighborhood operators, (ii) pivoting rules and (iii) bias towards good regions of the objective space. In this work, we conduct an extensive experimental campaign to analyze such factors in a Pareto local search (PLS) algorithm for the bi-objective bidimensional knapsack problem (bBKP). In the first set of experiments, we investigate PLS as a stand-alone algorithm, starting from random and greedy solutions. In the second set, we analyze PLS as a post-optimization procedure.

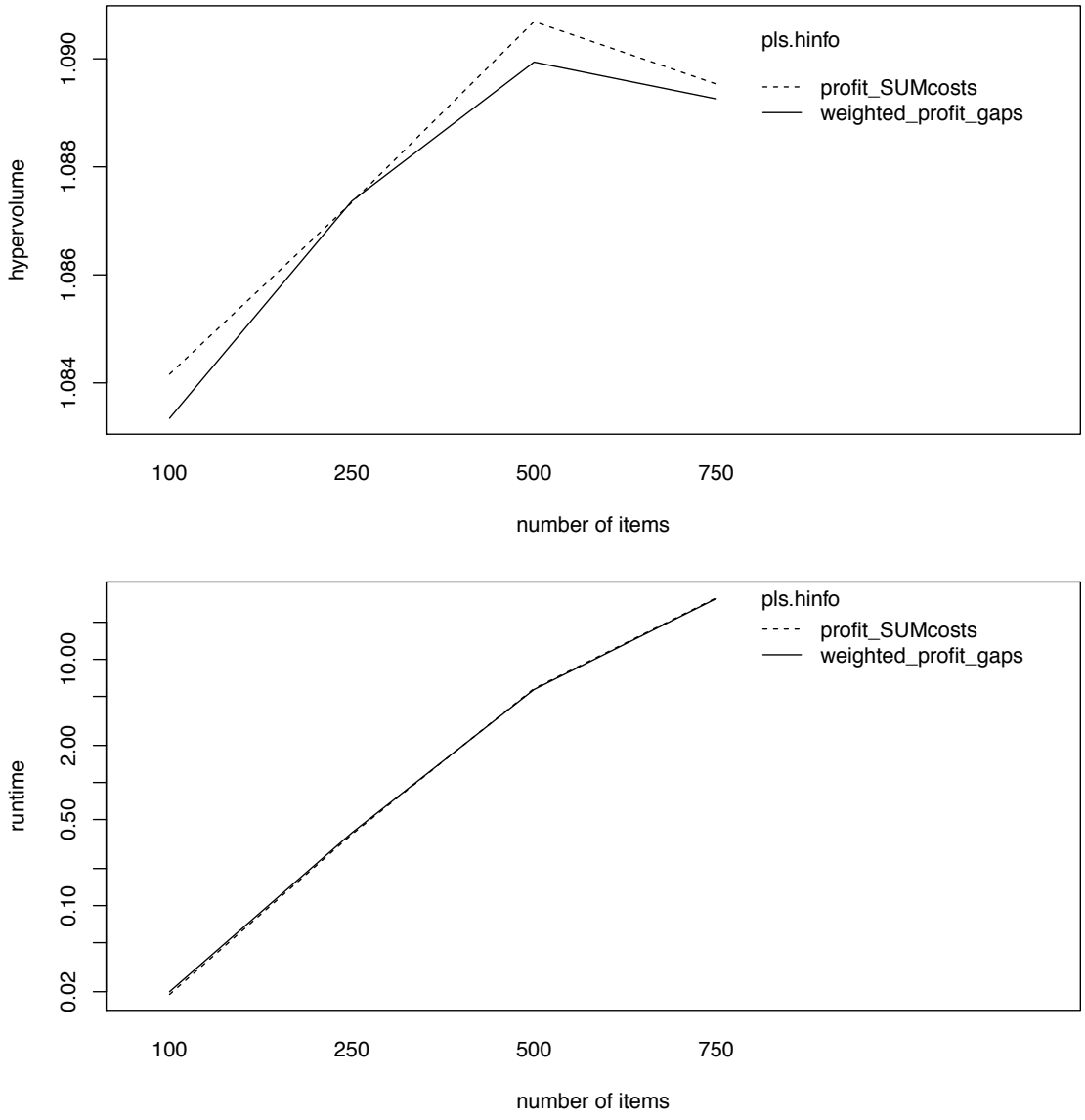

Fig. 1: Median hypervolume (top) and runtime (bottom) of different heuristics for PLS starting from greedy solutions and $r = 1$.

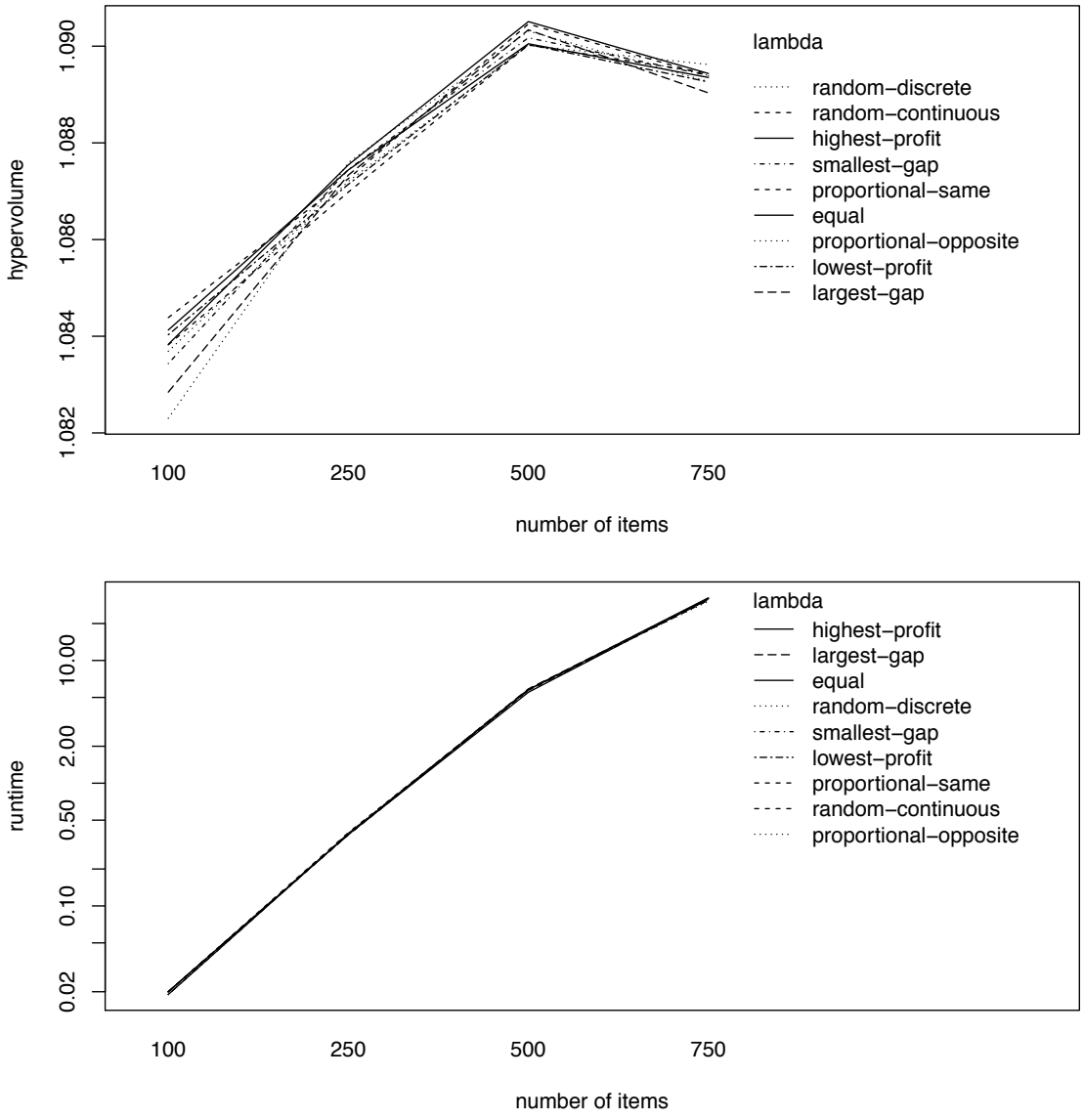

Fig. 2: Median hypervolume (top) and runtime (bottom) of different lambda generation methods for PLS starting from greedy solutions and $r = 1$.

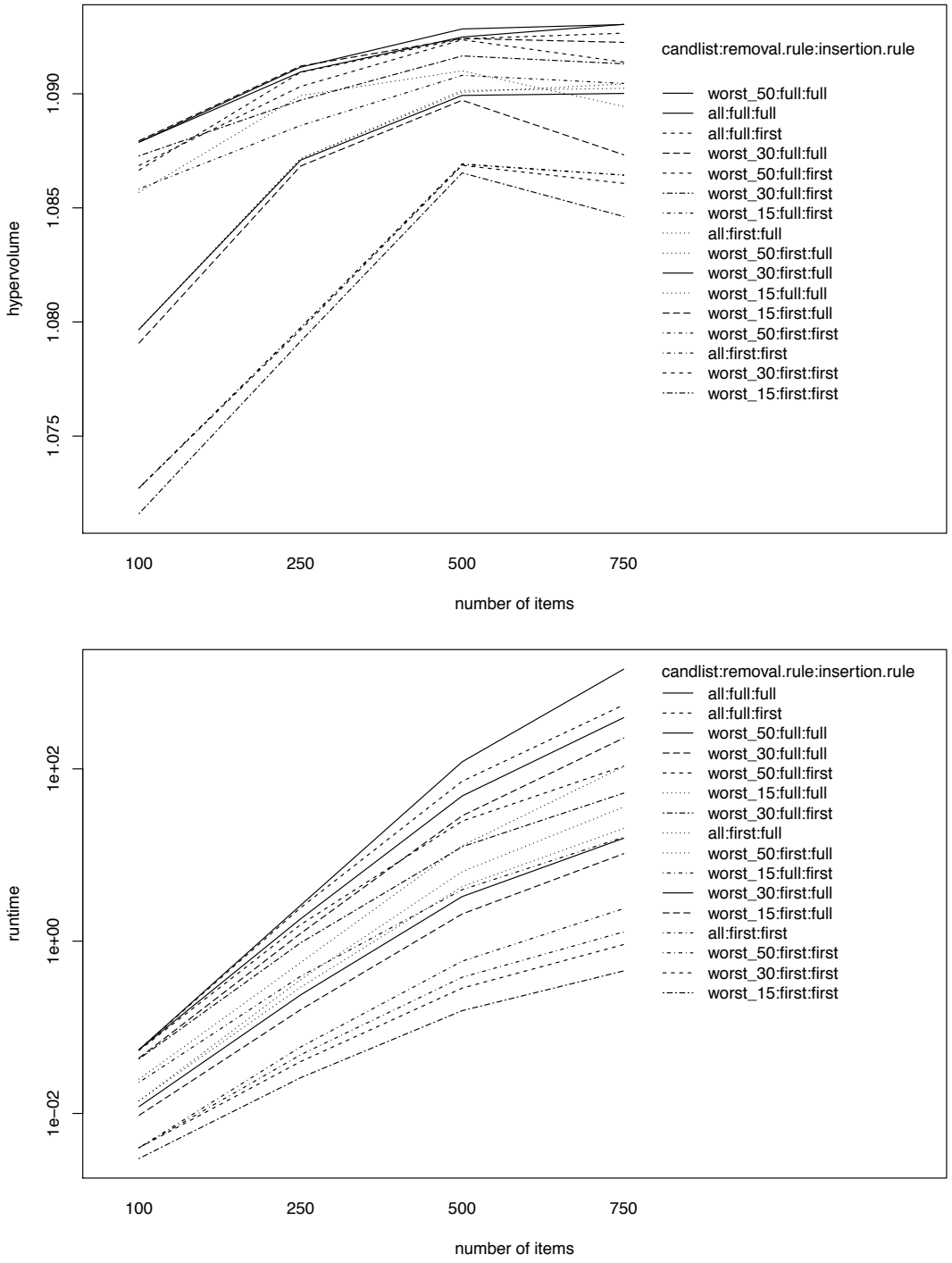

Fig. 3: Median hypervolume (top) and runtime (bottom) of different candidate list options and pivoting rules for PLS starting from greedy solutions and $r = 1$.

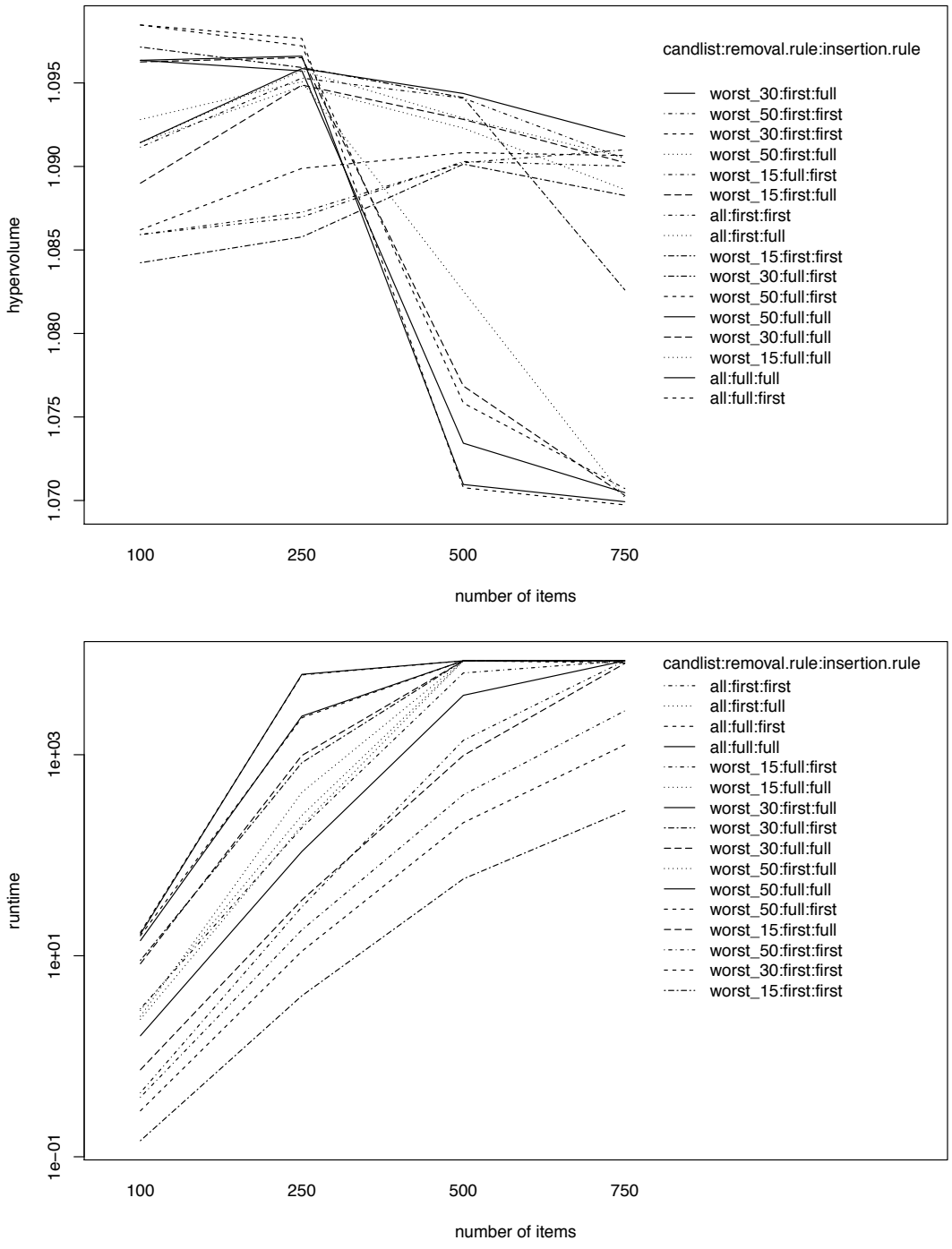

Fig. 4: Median hypervolume (top) and runtime (bottom) of different candidate list options and pivoting rules for PLS starting from greedy solutions and $r = 2$.