Multi-objective Local Search Based on Decomposition

Bilel Derbel[1], Arnaud Liefooghe[1(✉)], Qingfu Zhang[2], Hernan Aguirre[3], and Kiyoshi Tanaka[3]

[1] University Lille, CNRS, UMR 9189 – CRIStAL/Inria Lille-Nord Europe, Villeneuve-d'ascq, France
bilel.derbel@univ-lille1.fr
[2] Computer Science Department, City University, Kowloon Tong, Hong Kong
[3] Faculty of Engineering, Shinshu University, Nagano, Japan

Abstract. It is generally believed that Local search (Ls) should be used as a basic tool in multi-objective evolutionary computation for combinatorial optimization. However, not much effort has been made to investigate how to efficiently use Ls in multi-objective evolutionary computation algorithms. In this paper, we study some issues in the use of cooperative scalarizing local search approaches for decomposition-based multi-objective combinatorial optimization. We propose and study multiple move strategies in the Moea/d framework. By extensive experiments on a new set of bi-objective traveling salesman problems with tunable correlated objectives, we analyze these policies with different Moea/d parameters. Our empirical study has shed some insights about the impact of the Ls move strategy on the anytime performance of the algorithm.

1 Introduction

Several single-objective approaches, ranging from problem-specific algorithms to more generic approaches such as meta-heuristics and evolutionary algorithms, have been designed, tuned and studied extensively in combinatorial optimization. Among many others, local search (Ls) heuristics [2] refer to algorithms where a solution is improved in an iterative search process by performing little perturbation on its vicinity. A common ingredient being at the basis of this class of algorithms is the so-called *neighborhood exploration* and *move strategy*. The specification of at least one neighborhood structure and its proper combination with a move strategy is in general a cornerstone in the design of advanced single-objective Ls-based algorithms. Actually, this statement holds also when turning to the multi-objective setting, where a whole set of solutions, optimizing simultaneously two or more objective functions, is to be computed. Ls components have been investigated to design effective aggregation-based [3,4,10,12] and dominance-based [9,10,12] multi-objective algorithms. In particular, within the class of dominance-based algorithms, it is shown in [9] how different move strategies can have a deep impact on search performance. In this paper, we are interested in studying the new opportunities offered by

J. Handl et al. (Eds.): PPSN XIV 2016, LNCS 9921, pp. 431–441, 2016.
DOI: 10.1007/978-3-319-45823-6_40

the so-called MOEA/D (multi-objective evolutionary algorithm based on decomposition) [14] framework in incorporating LS components. In fact, MOEA/D is a recently-proposed aggregation-based framework which was extensively studied for continuous problems. Interestingly, MOEA/D is a reference algorithm in multi-objective optimization, mainly due to its high flexibility in incorporating different search paradigms, and the high quality of the so-obtained algorithms. Nonetheless, very few investigations can be found on the proper incorporation of LS within MOEA/D for discrete domains. Some adaptations exist, but they are often based on genetic operators [1,11], and relatively few in-depth investigations [5,6] considering LS in MOEA/D were conducted against the large body of works in continuous domains.

In this paper, we provide a comprehensive study on incorporating basic LS move strategies into the MOEA/D framework. More precisely, our contribution is three-fold. Firstly, we revisit conventional single-objective move strategies and illustrate how they can be hybridized with MOEA/D. In particular, we highlight how the replacement flow of MOEA/D can be adapted to support such strategies. Secondly, we study the performance of the so-designed algorithms using a new set of bi-objective traveling salesman problem (TSP) instances with tunable objective correlations. Our thorough experimental analysis shows that different behaviors can be obtained depending on objective correlation, and more importantly on available budgets. Our findings are the byproduct of a running time analysis providing evidence on the importance of the LS move strategy in the design of anytime decomposition-based multi-objective algorithms. Thirdly, we provide a comprehensive study on the impact of MOEA/D common parameters. The research conducted in this paper is also to be viewed as establishing the first steps towards the design of more powerful decomposition-based multi-objective algorithms based on more advanced local search components. In fact, notwithstanding that we are not horse-racing against state-of-the-art algorithms for the considered optimization problems, and that we consider basic move strategies, our findings on the anytime performance of the designed algorithms suggests that incorporating LS into MOEA/D is still in its very infancy beginning, and hence, would deserve further research investigations in the future.

The rest of this paper is organized as follows. In Sect. 2, we recall some background on LS and MOEA/D. In Sect. 3, we describe in more details different strategies for incorporating LS components into MOEA/D. In Sect. 4, we give our experimental setup. In Sect. 5, we discuss our experimental findings. In Sect. 6, we conclude the paper and discuss some open research directions.

2 Background

A multi-objective optimization problem (MOP) can be defined by a solution set X and by an objective function vector $f = (f_1, \ldots, f_m)$ to be minimized.

The MOEA/D [14] **framework.** MOEA/D falls into the class of decomposition-based algorithms. It seeks good-performing solutions in multiple regions of the Pareto front by *decomposing* the original MOP into a number of *scalarized*

single-objective *sub-problems*. Different scalarizing functions have been proposed so-far. In this paper, we use the common weighted Chebyshev function, to be minimized: $g(x \mid \lambda, z^\star) = \max_{k\in\{1,\ldots,m\}} \lambda_k \cdot \left|z_k^\star - f_k(x)\right|$; where $x \in X$, $\lambda = (\lambda_1, \ldots, \lambda_m)$ is a positive weighting coefficient vector, and $z^\star = (z_1^\star, \ldots, z_m^\star)$ is a reference point. In this respect, the originality of the MOEA/D framework is to define a *T-neighborhood relation* between sub-problems. Let $(\lambda^1, \ldots, \lambda^\mu)$ be a set of μ uniformly distributed weighting coefficient vectors defining μ sub-problems. MOEA/D maintains a population $P = (x^1, \ldots, x^\mu)$, where every individual corresponds to one sub-problem. For each sub-problem $i \in \{1, \ldots, \mu\}$, its T-neighbors, denoted $\mathcal{B}(i)$, are defined by considering the T closest weight vectors. Sub-problem solutions are evolved with respect to their neighbors. For every sub-problem, an offspring solution from the T-neighbors set $\mathcal{B}(i)$ is generated using some evolutionary operators. Then, the offspring can replace one or more T-neighbors if it improves the scalar (Chebyshev) value of the corresponding solution of the neighboring sub-problem. Different variants of this baseline MOEA/D flow exist. In the remainder, we consider the modifications introduced in [8], considered as a state-of-the-art variant in continuous domains, where (i) the T-neighbors of a sub-problem is the whole population with a small probability δ, or $\mathcal{B}(i)$ otherwise, and (ii) a newly generated offspring can replace at most nr other solutions, where nr and δ are two user-defined parameters. Other MOEA/D variants could be considered as well, but for the sake of analysis, we only consider the most common and widely-used variant from [8,14].

LS **Move Strategies.** LS is a single solution-based walk that iteratively improves the current solution by means of local transformations, and then moving to an improving close-by solution. Those transformations are usually based on a *neighborhood* function $\mathcal{N} : X \to 2^X$, which assigns a set of neighboring solutions $\mathcal{N}(x) \subset X$ to any solution $x \in X$. It should be clear for the reader that we differentiate between the T-neighborhood of MOEA/D and the neighborhood of a solution in LS. In the most simple LS variant, also referred to as *hill-climbing*, the search stops when the current solution is not outperformed by any neighbor. This means that a *local optimum* is reached. The move strategy, defining the transition rule to select an improving neighbor, is also a key ingredient in LS-based search. Typical strategies are as follows: (i) In a *best-improvement* (or steepest descent) move, the neighbor that improves the most is selected at each iteration. This means that the whole neighborhood is generated, which can be time-consuming for large neighborhoods. (ii) In a *first-improvement* move, the first improving neighbor is immediately selected. This avoids to systematically generate and evaluate the whole neighborhood. The exploration order of neighbors can remain unchanged, or instead can be randomly shuffled at each iteration. Additionally, the neighborhood structure can be used as a an evolutionary mutation operator when some few neighboring solutions are sampled at random. Hence, (iii) a *random* strategy can be considered as well, where a random neighbor is generated and replaces the current solution if there is an improvement.

3 The MLSD Scheme

Incorporating LS into MOEA/D can be viewed as a natural outcome since several single-objective sub-problems are to be improved cooperatively. Although the standard neighborhood exploration mechanisms of LS might not be very complicated to integrate into MOEA/D, still important design technicalities have to be explicitly and carefully specified, especially when exploring new neighboring solutions and when performing replacement in original MOEA/D.

In the high-level pseudo-code depicted in Algorithm 1, we provide a relatively detailed description of different possible ways of hybridizing MOEA/D with LS move policies. The proposed scheme is called MLSD-SR (Multi-objective Local Search based on Decomposition). One should notice that MLSD is parametrized by two elements, namely s (referring to the *selection* policy) and r (referring to the *replacment* policy). This allows us to differentiate between two stages: (i) the move selection stage (lines 10 to 21), and (ii) the replacement stage (lines 22 to 29). We thereby obtain four possible variants, as discussed in the following.

Algorithm 1. MLSD-SR: high-level pseudo-code

Input: μ: population size; T: neighborhood size; $\delta \in [0,1]$; $nr \in [\![0, \mu]\!]$; s $\in$ {Best, First, Rnd}; r $\in$ {Min, Rnd}.

1 $\{\lambda^1, \ldots, \lambda^\mu\} \leftarrow$ generate weight vectors w.r.t. μ sub-problems;
2 $\forall i \in \{1, \ldots, \mu\}\ \mathcal{B}(i) \leftarrow$ the T closest sub-problems w.r.t λ_i;
3 $P = \{x^1, \ldots, x^\mu\} \leftarrow$ generate the initial population;
4 **evaluate** P;
5 (**update** external archive with P;) /* optional */
6 **set** $z^\star$ from P;
7 **while** STOPPING CONDITION **do**
8 **for** $i \in \{1, \ldots, \mu\}$ **do**
9 **if** *rand*$\{[0,1]\} < \delta$ **then** $B_i \leftarrow \mathcal{B}(i)$; **else** $B_i \leftarrow P$;
  // Stage #1: Move selection
10 $k \leftarrow$ **rand** $\{B_i\}$;
11 $I \leftarrow \emptyset$;
  /* Check moves and record improved sub-problems */
12 **for** $y \in \mathcal{N}(x^k)$ **do** /* By default, s = Best */
13 **evaluate** y;
14 (**update** external archive with y;) /* optional */
15 **update** $z^\star$ using y;
16 $J_y \leftarrow \{j \in B_i \text{ s.t. } g(y \mid \lambda^j, z^\star) < g(x^j \mid \lambda^j, z^\star)\}$;
17 **if** $J_y \neq \emptyset$ **then**
18 $c_y \leftarrow 0$;
19 $I \leftarrow I \cup \{(y, c_y, J_y)\}$;
20 **if** *s = First* **then break**;
21 **if** *s = Rnd* **then break**; /* go to line 22 */
  // Stage #2: Replacement
22 **while** $\exists j \in B_i$ *s.t.* $(\exists (y, c_y, J_y) \in I$ *s.t.* $j \in J_y$ *and* $c_y < nr)$ **do**
23 **if** *r = Min* **then**
24 $y^* \leftarrow \mathbf{arg\,min}_{y \text{ s.t. } (y, c_y, J_y) \in I} \left\{ g(y \mid \lambda^j, z^\star) \right\}$
25 **else if** *r = Rnd* **then**
26 $y^* \leftarrow$ **rand** $\{y \text{ s.t } (y, c_y, J_y) \in I\}$;
27 $x^j \leftarrow y^*$;
28 $c_{y^*} \leftarrow c_{y^*} + 1$;
29 $B_i \leftarrow B_i \setminus \{j\}$;

The MLSD scheme iteratively loops over sub-problems until a stopping condition is satisfied. At each iteration w.r.t. sub-problem i, two stages are performed. The first stage consists in generating some new candidate solutions to be considered in the second stage. First, a parent solution x^k is selected randomly from the neighborhood of sub-problem i. The selected solution is then locally explored using the LS neighborhood structure $\mathcal{N}$. Three different move strategies can be considered. The first one ($\mathsf{s} = \mathsf{Best}$) consists in traversing all solutions $y \in \mathcal{N}(x^k)$ in an exhaustive manner while checking for any improvement. Notice that variable J_y (line 16) denotes the set of sub-problems improved by an incumbent solution y, and c_y is a counter initialized to 0. The tuple (y, c_y, J_y) is then saved into set I which contains all the records w.r.t any improving solution in $\mathcal{N}(x^k)$. In the second strategy ($\mathsf{s} = \mathsf{First}$), the exploration of neighbors $\mathcal{N}(x^k)$ stops as soon as an improving solution y is found. This strategy guarantees that if $\mathcal{N}(x^k)$ contains at least one improving solution, then it is selected and recorded in set I for the next stage. The last move strategy ($\mathsf{s} = \mathsf{Rnd}$) picks a single incumbent solution y uniformly at random from $\mathcal{N}(x^k)$, and records the tuple (y, c_y, J_y) in set I only if y is improving at least one neighboring sub-problem.

The second stage consists in replacing the solutions of neighboring sub-problems. If no improvement was observed, then the replacement stage is simply skipped. Otherwise, i.e. when $|I| \geq 1$, two possible strategies are considered. In the first one ($\mathsf{s} = \mathsf{Min}$), the solution of every sub-problem j in the T-neighborhood of sub-problem i is replaced by the best improving solution $y^\star$ found during the previous stage (if any). In the second one ($\mathsf{s} = \mathsf{Rnd}$), an improving solution (if any) is picked randomly to replace the current solution of j. Notice that in case the set I contains one single recorded tuple, the two previous replacement strategies are equivalent. Notice also that if a First or a Rnd policy is adopted in the selection stage, the designed replacement strategies are also equivalent. Hence, the two replacement strategies might imply different variants of MLSD only when a Best strategy is adopted in the first stage.

Finally, it is important to notice the role of the nr parameter in the replacement stage. In fact, since several candidate improving solutions can be considered in the case $\mathsf{s} = \mathsf{Best}$, each time a solution y is selected for the replacement in line 27, its associated counter c_y is incremented. Consequently, once this counter reaches the value nr, the corresponding solution cannot be selected anymore to replace any sub-problem, as specified by the condition of line 22.

4 Experimental Setup

For the sake of studying the behavior of the MLSD-SR framework, we consider the Traveling Salesman Problem (TSP) as a baseline benchmark problem. The motivation behind this choice is two fold. First, permutation-based optimization problems, like TSP, are of choice when evaluating the behavior of LS-based algorithms. Second, the TSP is a fundamental problem that appears at the bottleneck of many real-world applications and is representative of a wide range of more complex combinatorial optimization problems. We emphasize that this choice is to be understood from a purely benchmarking perspective. In particular, it is worth noticing

that the multi-objective TSP has attracted a lot of interest in recent years and one can report several state-of-the-art algorithms, see e.g. [5,9,10,12]. This paper does not propose yet another algorithm for TSP, and we shall not consider to compare the MLSD-SR with those algorithms. Besides, designing TSP-specific algorithms is a whole piece of research that we are not targeting in this experimental study. Accordingly, we shall only focus on analyzing the relative performance of the different move strategies described previously.

Multi-objective TSP with Correlated Objectives. Given a complete graph $G = (V, E)$ with n nodes and non-negative edge costs, the symmetric single-objective TSP seeks a cyclic permutation that contains each node exactly once and such that the total cost is minimized. A solution can be represented as a permutation π of size n. Since multiple costs like distance or travel time can be considered, a multi-objective variant of the TSP can be formulated. Let $\{v_1, v_2, \ldots, v_n\}$ be the set of nodes, and $\{[v_i, v_j] \,|\, v_i, v_j \in V\}$ the set of edges. In the m-objective case, we have m cost matrices such that each edge $[v_i, v_j] \in E$ is assigned a *cost* c_{ij}^k for each objective function $k \in \{1, \ldots, m\}$. The objective functions can then be defined as follows: $f_k(\pi) = c^k_{\pi(n)\pi(1)} + \sum_{i=1}^{n-1} c^k_{\pi(i)\pi(i+1)}$. The multi-objective TSP is known to be NP-hard and intractable [10]. In this paper, we consider two-objective symmetric TSP instances ($m = 2$) with *correlated* random distance matrices. Following [12], edge costs are chosen from a uniform distribution in $[0, 4473]$. However, we additionally define a correlation coefficient $\rho \in [-1, 1]$ between the data contained in both cost matrices. The generation of correlated data follows a multivariate uniform distribution [13]. The positive (resp. negative) data correlation allows to decrease (resp. increase) the degree of conflict between the objective function values with a high accuracy. Notice than when $\rho = 0$, our instances are the same as [12].

Parameter Setting. We consider the *2-opt exchange* operator as the *neighborhood* $\mathcal{N}$ for TSP, i.e. given a candidate solution π, the sequence of nodes located between $\pi(i)$ and $\pi(j)$ is reversed. The neighborhood size is hence $\frac{n \cdot (n-1)}{2}$. We experiment instances of size $n = 100$ and correlation values: $\rho \in \{$-0.8, -0.4, 0.0, 0.4, 0.8$\}$. We consider a broad range for the other parameters, namely population size $\mu \in \{50, 100, 150, 200\}$, T-neighborhood size $T \in \{5, 10, 15, 20\}$, $nr \in \{1, 2, \infty\}$, and $\delta \in \{0.0, 0.1\}$. For every parameter combination, we consider the four variants of MLSD-SR as summarized in the table below, thus ending up with 1 920 configurations, each one independently executed 20 times. For s = First, neighboring solutions are explored in a random order. The stopping condition is a maximum budget of 10^8 function evaluations. The initial population is generated randomly and the weight vectors are generated as in [14].

r \ s	Best	First	Rnd
Rnd	✓(MLSD-BR)		
Min	✓(MLSD-BM)	✓(MLSD-FM)	✓(MLSD-RM)

5 Experimental Analysis

We follow the performance assessment protocol proposed in [7] by using the hypervolume relative deviation (I_{hv}) and the additive epsilon (I_{ε}^{+}) indicators. The hypervolume reference point is set to the worst objective-value, and the reference set is the best-found approximation over all tested configurations. Notice that we use an external archive recording all non-dominated solutions found so far.

High Budget Setting. We first report the descriptive statistics on the indicator-values, together with a Mann-Whitney non-parametric statistical test with a p-value of 0.05 and using a Bonferroni correction, for the highest budget of 10^8 calls of the evaluation function. In Table 1, we show the rank of different MLSD-SR variants with the rank being the number of variants that statistically outperform the one under consideration for each instance. The lower the rank, the better the algorithm. Both indicators agree that the best performing variant of MLSD over all considered instances is when a Best move strategy is adopted together with a Min replacement strategy. The objective correlation of considered instances appear to have a crucial impact. The gap between MLSD-BM and the other variants is substantial in the case of conflicting objectives whereas we found no significant differences for highly correlated objectives. Overall, the considered MLSD variants can be ranked as follows: MLSD-BM $>$ MLSD-BR $\approx$ MLSD-FM $>$ MLSD-RM. It is important to remark that combining a Best move strategy with an elitist replacement strategy is crucial, otherwise a First move strategy would be more appropriate. Notice that at this stage of the analysis, the MLSD-RM variant is overall the worst performing one, and the relative performance gap between different T-neighborhoods are not statistically significant. In the following, we shall show that these preliminary conclusions can only hold for a high computational budget.

Anytime Analysis. When analyzing the quality of the approximation with different budgets, we basically find that the relative performance of the considered variants is deeply impacted, independently of the parameter setting. This is illustrated in Fig. 1 for a particular parameter setting. Interestingly, the MLSD-BM and MLSD-BR variants can only outperform the other variants for a high budget. MLSD-RM, which was shown to be the worst-performing approach in such a setting, now appears to be the best anytime strategy. This might be surprising at a first glance. However, in the early stages of the search process, it is more likely that among few random samples, an improving solution for different sub-problems is found. In contrast, MLSD-BM would anyway explore all neighboring solutions (quadratic in n) and consider at most one solution for replacement. Hence, MLSD-RM is likely to progress faster and to save a significant number of evaluations. As the quality of the population gets better, it becomes more unlikely to find improving neighbors using random sampling. This can explain why MLSD-RM gets stuck and cannot improve the quality of the population anymore. It is also interesting to remark that MLSD-FM provides an intermediate trade-off, since it is relatively competitive against MLSD-RM while being able to catch MLSD-BM again on the latest stages. Interestingly, these

Table 1. Algorithm rank summary using 10^8 function evaluations, $\mu = 100$, $nr = 2$ and $\delta = 0.1$. The number in brackets stands for the average indicator-value.

		Hypervolume relative deviation ($I_{\text{hv}} \cdot 10^{-2}$)				Additive epsilon indicator ($I_\varepsilon^+ \cdot 10^2$)			
		s = B		MLSD-FM	MLSD-RM	s = B		MLSD-FM	MLSD-RM
ρ	T	MLSD-BM	MLSD-BR			MLSD-BM	MLSD-BR		
−0.8	5	0 (1.41)	4 (2.07)	4 (1.95)	12 (2.61)	0 (49.45)	5 (75.43)	4 (66.89)	5 (78.23)
	10	0 (1.38)	4 (2.05)	4 (2.02)	12 (2.57)	0 (51.21)	5 (85.53)	5 (86.38)	5 (76.68)
	15	0 (1.33)	4 (1.98)	6 (2.17)	12 (2.57)	0 (52.27)	5 (82.10)	10 (91.86)	5 (76.72)
	20	0 (1.39)	4 (2.04)	10 (2.28)	12 (2.47)	0 (53.95)	6 (86.20)	14 (103.3)	5 (77.53)
−0.4	5	0 (1.83)	1 (1.95)	8 (2.22)	12 (2.64)	0 (50.63)	2 (58.36)	4 (66.92)	8 (72.60)
	10	0 (1.78)	0 (1.84)	2 (2.03)	12 (2.50)	0 (50.92)	2 (60.35)	4 (65.97)	6 (68.70)
	15	0 (1.70)	0 (1.92)	5 (2.08)	12 (2.56)	0 (49.39)	2 (58.69)	6 (68.77)	8 (71.54)
	20	0 (1.78)	1 (1.95)	5 (2.06)	12 (2.51)	0 (52.14)	3 (60.60)	6 (69.52)	7 (69.94)
0.0	5	0 (2.42)	0 (2.30)	5 (2.67)	1 (2.69)	0 (45.08)	0 (41.62)	4 (51.59)	4 (52.27)
	10	0 (2.23)	0 (2.28)	0 (2.44)	5 (2.85)	0 (39.84)	0 (41.71)	0 (47.41)	6 (52.98)
	15	0 (2.32)	0 (2.25)	0 (2.52)	7 (2.71)	0 (42.15)	0 (42.22)	0 (49.12)	7 (50.31)
	20	0 (2.39)	0 (2.26)	0 (2.49)	7 (2.80)	0 (43.79)	0 (41.02)	0 (47.95)	7 (53.25)
0.4	5	0 (2.66)	0 (2.33)	0 (2.61)	0 (2.47)	1 (44.82)	0 (38.06)	0 (42.65)	0 (40.59)
	10	0 (2.51)	0 (2.43)	0 (2.44)	0 (2.50)	0 (42.17)	0 (39.45)	0 (38.80)	0 (39.44)
	15	0 (2.59)	0 (2.34)	0 (2.54)	0 (2.64)	0 (39.49)	0 (37.86)	0 (42.62)	0 (42.86)
	20	0 (2.54)	0 (2.30)	0 (2.68)	0 (2.52)	0 (39.23)	0 (38.48)	0 (42.14)	0 (41.33)
0.8	5	0 (2.54)	0 (2.15)	0 (2.08)	0 (2.10)	0 (33.76)	0 (29.78)	0 (28.00)	0 (28.25)
	10	0 (2.49)	0 (2.21)	0 (2.05)	0 (2.36)	0 (32.83)	0 (30.17)	0 (28.21)	0 (31.87)
	15	0 (2.56)	0 (2.22)	0 (2.14)	0 (2.31)	0 (32.78)	0 (28.68)	0 (27.56)	0 (30.62)
	20	0 (2.39)	0 (2.40)	0 (2.23)	0 (2.16)	0 (31.57)	0 (31.39)	0 (29.60)	0 (28.54)

results suggest that there is much room for future improvements in the anytime behavior of MLSD by considering hybrid move strategies.

Impact of the Population Size (μ). In Fig. 2, we show a subset of results on the impact of different population sizes on MLSD-BM and MLSD-RM (since no significant impact was found for MLSD-FM). The larger the population size, the better the final approximation set, independently of the considered strategy.

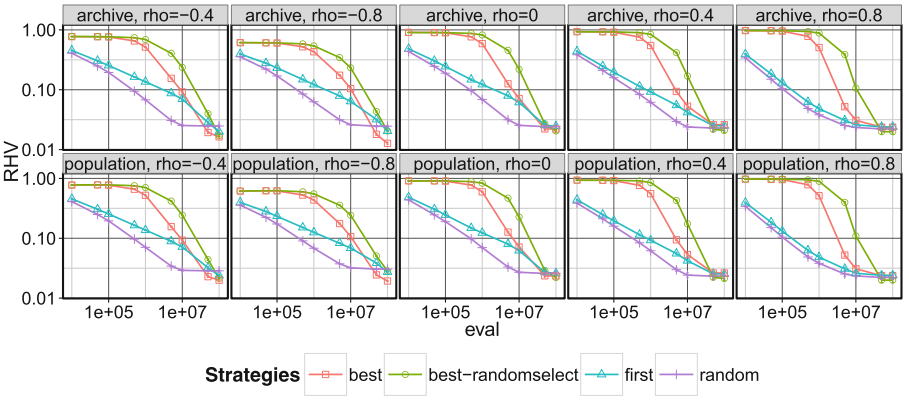

Fig. 1. Runtime analysis of the different algorithm variants. Error bars indicate 95 % confidence intervals. $\delta = 0$, $T = 10$, $nr = \infty$ and $\mu = 100$. Notice the log-scales.

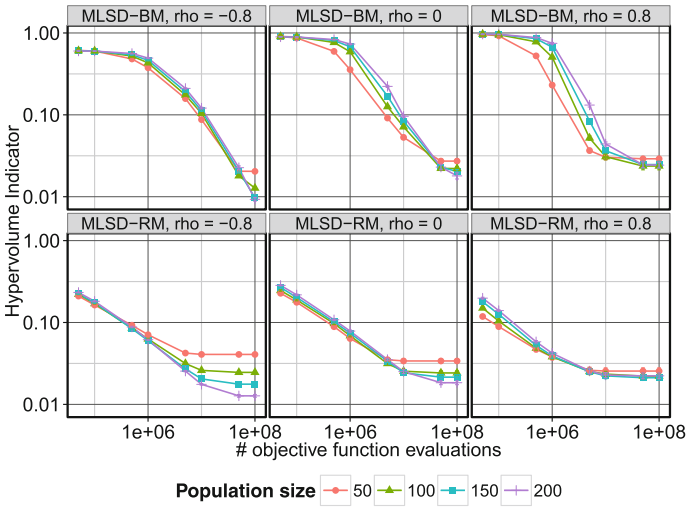

Fig. 2. Runtime analysis for different population sizes. $\delta = 0$, $T = 10$, $nr = \infty$.

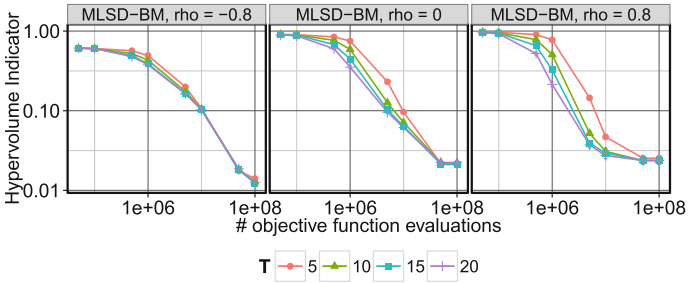

Fig. 3. Runtime analysis for different $T-$values. $\delta = 0$, $nr = \infty$ and $\mu = 100$.

However, smaller population sizes are better for smaller budgets, especially for instances with correlated objectives. We attribute this to the fact that a larger population size impacts the population diversity, and is thus more critical when the Pareto front is larger, which is the case for conflicting objectives.

Diversity Issues (T, nr and δ). We are able to report a significant impact of the T-neighborhood size only for the MLSD-BM variant, for highly correlated objectives and a small budget, as illustrated in Fig. 3. As for parameter nr, we found a significant impact only for MLSD-FM and MLSD-RM, as illustrated in Fig. 4. We recall that a larger $nr-$value allows a high-quality solution, possibly improving multiple sub-problems simultaneously, to replace all those solutions at once. Intuitively, the surviving solution has then more chance to improve the overall population quality in subsequent iterations, but at the price of decreasing diversity. we can see that smaller $nr-$values are better for convergence purposes,

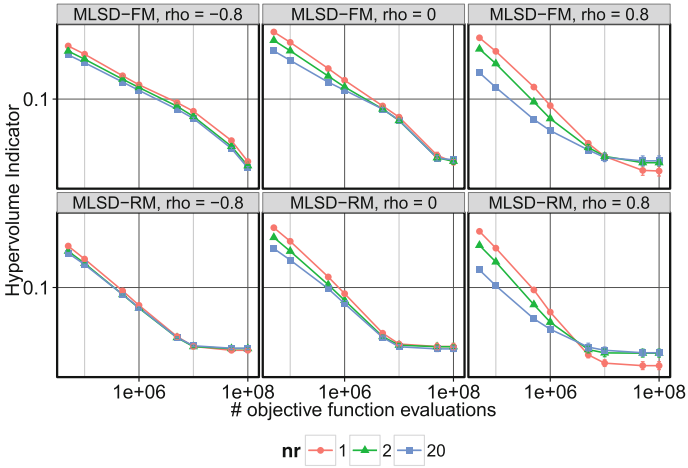

Fig. 4. Runtime analysis for different $nr-$values. $\delta = 0$, $T = 10$ and $\mu = 100$.

whereas a larger $nr-$value provides a better performance for small budgets. Interestingly, this observation holds only for highly-correlated objectives. As for parameter δ, the impact on performance was only significant when using MLSD-BM for correlated objectives with a small T-neighborhood size, but it was not helpful for improving the relative anytime performance. These empirical observations suggest that, contrary to the continuous case, the δ parameter might not be of great help when tackling combinatorial problems with conflicting objectives.

6 Conclusion

This paper investigates the foundations of the design of cooperative scalarizing local search approaches within decomposition-based algorithms for multi-objective combinatorial optimization. Our results revealed strong evidence on the need of adaptive algorithms that would enable to mix different move strategies and to better combine the neighborhood exploration with the replacement stage in order to properly balance the exploration/exploitation trade-off. It is our hope that our empirical study can enlighten our current understandings of decomposition-based approaches for multi-objective combinatorial optimization, and can stimulate new research paths towards the design of more powerful multi-objective randomized search heuristics based on local search and decomposition.

References

1. Chang, P.C., Chen, S.H., Zhang, Q., Lin, J.L.: MOEA/D for flowshop scheduling problems. In: CEC, pp. 1433–1438 (2008)
2. Hoos, H., Stützle, T.: Stochastic Local Search: Foundations and Applications. Morgan Kaufmann, Burlington (2004)

3. Ishibuchi, H., Murata, T.: A multi-objective genetic local search algorithm and its application to flowshop scheduling. IEEE Trans. Cyber. **28**(3), 392–403 (1998)
4. Jaszkiewicz, A.: Genetic local search for multi-objective combinatorial optimization. EJOR **137**(1), 50–71 (2002)
5. Ke, L., Zhang, Q., Battiti, R.: MOEA/D-ACO: a multiobjective evolutionary algorithm using decomposition and ant colony. IEEE Trans. Cyber. **43**(6), 1845–1859 (2013)
6. Ke, L., Zhang, Q., Battiti, R.: Hybridization of decomposition and local search for multiobjective optimization. IEEE Trans. Cyber. **44**(10), 1808–1820 (2014)
7. Knowles, J., Thiele, L., Zitzler, E.: A tutorial on the performance assessment of stochastic multiobjective optimizers. TIK report 214, Zurich, Switzerland (2006)
8. Li, H., Zhang, Q.: Multiobjective optimization problems with complicated Pareto sets, MOEA/D and NSGA-II. IEEE TEC **13**(2), 284–302 (2009)
9. Liefooghe, A., Mesmoudi, S., Humeau, J., Jourdan, L., Talbi, E.G.: On dominance-based local search. J. Heuristics **18**(2), 317–352 (2012)
10. Lust, T., Teghem, J.: Two-phase Pareto local search for the biobjective traveling salesman problem. J. Heuristics **16**(3), 475–510 (2010)
11. Palacios Alonso, J.J., Derbel, B.: On maintaining diversity in MOEA/D: application to a biobjective combinatorial FJSP. In: GECCO, pp. 719–726 (2015)
12. Paquete, L., Stützle, T.: Design and analysis of stochastic local search for the multiobjective traveling salesman problem. COR **36**(9), 2619–2631 (2009)
13. Verel, S., Liefooghe, A., Jourdan, L., Dhaenens, C.: On the structure of multi-objective combinatorial search space: MNK-landscapes with correlated objectives. Eur. J. Oper. Res. **227**(2), 331–342 (2013)
14. Zhang, Q., Li, H.: MOEA/D: a multiobjective evolutionary algorithm based on decomposition. IEEE TEC **11**(6), 712–731 (2007)