Queued Pareto Local Search for Multi-Objective Optimization

Maarten Inja, Chiel Kooijman, Maarten de Waard,
Diederik M. Roijers, Shimon Whiteson
{maarten.inja, chiel.kooijman, mrtndwrd}@gmail.com,
{d.m.roijers, s.a.whiteson}@uva.nl

Informatics Institute, University of Amsterdam

Abstract. Many real-world optimization problems involve balancing multiple objectives. When there is no solution that is best with respect to all objectives, it is often desirable to compute the *Pareto front*. This paper proposes *queued Pareto local search* (QPLS), which improves on existing *Pareto local search* (PLS) methods by maintaining a queue of improvements preventing premature exclusion of dominated solutions. We prove that QPLS terminates and show that it can be embedded in a genetic search scheme that improves the approximate Pareto front with every iteration. We also show that QPLS produces good approximations faster, and leads to better approximations than popular alternative MOEAs.

1 Introduction

Many real-world optimization problems contain multiple objectives, e.g., when mapping different processes of a software application to hardware components, both processing time and power consumption should be minimized [7].

For specialized applications, it is sometimes possible to compute an exact set of optimal trade-offs between the objectives, i.e., the Pareto front. However, when the solution space of these problems is large, computing the Pareto front is often intractable. In this case, *multi-objective evolutionary algorithms* (MOEAs) [2] can compute a set of solutions that approximates the Pareto front. MOEAs are general-purpose multi-objective optimization methods, and have been applied to a large variety of optimization problems [1], from reinforcement learning [15] to design space exploration [12].

To speed up MOEAs, one can use *Pareto local search* (PLS) algorithms [5, 9, 13]. PLS algorithms employ simple heuristic improvement algorithms to find reasonably good solutions in a short time. PLS methods find an approximate Pareto front by looking in the (search space) neighborhood of the individual solutions in this archive for solutions that improve upon the current Pareto archive. An important advantage of this method is that the size of the archive is not limited. This is important because the size of the Pareto front is often not known in advance.

However, an important downside of current PLS methods is that promising solutions are excluded from the Pareto-archive when an improvement is found for another (unrelated) solution, before they have a chance to improve themselves.

Such premature deletions from the archive are undesirable since they reduce genetic diversity that might be needed to find part of the Pareto front.

In this paper, we propose a new algorithm for Pareto local search that we call *queued Pareto local search* (QPLS), which stores promising solutions in a queue. When such a solution is popped off the queue, QPLS performs strict Pareto improvements by looking for a stricly better solution (in all objectives) in the neighborhood of the solution. Only when such improvements are no longer possible, is it compared to a Pareto archive. This prevents premature deletion of solutions. Unlike other PLS methods, the queued approach "protects" dominated solutions until they are mutated to a locally optimal state. As in previous PLS work, we embed QPLS in a genetic scheme.

We empirically compare Genetic QPLS against the state-of-the-art PLS method Genetic SPLS [5], and against the popular non-PLS MOEAs NSGA-II [3] and SPEA2 [21], using multi-objective coordination graphs [16, 17]. We show that QPLS can produce good approximate fronts for much larger problems than that can be solved with exact methods (such as [18]). Furthermore, we show that Genetic QPLS maintains a Pareto set with a larger *hypervolume* [19] than the other algorithms both in the short and long run.

2 Background

We first introduce the problem setting and notation used throughout the paper. A *multi-objective optimization problem* (MOOP) is a tuple $< \mathcal{V}, \mathcal{S}, \mathbf{f} >$, where:

- $\mathcal{V} = \{v_1, ..., v_n\}$ is the set of n enumerated *variables*,
- $\mathcal{S} = \mathcal{S}_1 \times ... \times \mathcal{S}_n$ is the *search space*, i.e., the Cartesian product of the domains of the variables in $\mathcal{V}$, and
- $\mathbf{f} : \mathcal{S} \rightarrow \Re^d$ is a d-objective fitness function that maps all points in the search space to a d-objective fitness. We refer to the value of the i-th objective as f_i.

We define the neighborhood of s, $\mathcal{N}(s)$, as the set of solutions that differ in exactly one variable. We assume that each objective needs to be maximized. We refer to an approximation to the Pareto front as a *Pareto archive*, denoted P. A solution s *Pareto dominates* a solution s' when its fitness is larger in one objective, and at least equal in all the others: $\mathbf{f}(s) \succ \mathbf{f}(s') = ((\forall i)\ f_i(s) \geq f_i(s'))\ \wedge\ ((\exists j)\ f_j(s) > f_j(s'))$. In a Pareto archive, there are no solutions that are Pareto dominated by another solution in that set. We say that a solution s *weakly dominates* another solution s', when it either dominates it, or has equal value. If s does not dominate s' and s' does not dominate s then s and s' are *incomparable*. Two Pareto archives P and P' can be *merged* [5] by taking the union and removing the dominated solutions.

2.1 Non-PLS Methods

Many MOEAs (e.g., [3, 21]) do not employ PLS but instead maintain a population of individual solutions that are improved upon via mutation and crossover operators. When these operators are ergodic, the entire search space is scanned in the limit [6]. This guarantees that MOEAs do not get permanently stuck in local optima. This holds even for MOEAs that do not employ PLS.

Two popular MOEAs that do not employ PLS are NSGA-II [3] and SPEA2 [21]. NSGA-II applies elitism, uses a fast sorting algorithm and focuses on maintaining diversity within the population. SPEA2 employs a measure of *strength* for a solution based on the number of solutions that dominate it, the number of solutions that are dominated by it, and the distance to the k-th nearest neighbor. The advantage of both is that their population diversity prevents them from getting stuck in local optima. However, neither of these algorithms employ heuristics to find better solutions more quickly. NSGA-II and SPEA2 are the two most popular algorithms for MOOPs [20].

2.2 PLS Methods

Paquete et al. [13] propose a *Pareto local search* (PLS) algorithm that maintains an archive of non-dominated solutions P. Each solution $s \in P$ is visited to search its neighborhood $\mathcal{N}(s)$ for new solutions. This is achieved by merging the entire neighborhood of s into P. A drawback is that the neighborhood of one solution can overwrite the whole archive (especially early on), while the neighborhoods of the overwritten solutions can contain solutions that would improve P even further. Drugan and Thierens [5] propose a variation in which improvements are made to the archive by adding a single improving solution with respect to P (Pareto-dominant or Pareto-incomparable) from the neighborhood $\mathcal{N}(s)$ of one solution $s \in P$ at a time. This reduces the probability that promising solutions in P are deleted after an improvement from a different solution. However, the removed dominated solutions might still have had possible improvements.

Liefooghe et al. [9] generalize PLS algorithms into several categories based on the current set selection for neighborhood scanning, exploration strategy, archiving method and stopping conditions. Two types of archiving methods are considered. An *unbounded* archive can be used to store the set of all non-dominated solutions and a *bounded* archive stores a subset of the non-dominated solutions. Some algorithms only store the dominating solutions as the archive is filled. Other algorithms employ diversity criteria, such as crowding distance or ε-dominance, to limit the size of the archive. However, all of these algorithms generate improvements from the neighborhood of current elements of the archive and directly delete members of the archive due to these improvements, even though these deleted elements could yield improvements if their neighborhoods were inspected.

Two distinct strategies can be employed to select the next improving solution s' from the neighborhood of s, $\mathcal{N}(s)$. In the *best-improvement* strategy the entire neighborhood $\mathcal{N}(s)$ is scanned for the best improvement. The selected improvement s' is guaranteed not to be dominated by another other solution in $\mathcal{N}(s)$. In the *first-improvement* strategy, the first solution s' in $\mathcal{N}(s)$ that improves s is returned immediately. Hansen and Mladenović [8] find that for single-objective LS, first-improvement usually leads to better empirical results. Liefooghe et al. [9] confirm this result for PLS algorithms, as do Drugan and Thierens [5] for SPLS.

3 QPLS

Our main contribution, *queued Pareto local search*, is given in Algorithm 1. QPLS prevents the premature deletion of promising solutions by maintaining a queue of solutions to improve, which leads to a more diverse Pareto archive.

Algorithm 1 QPLS($\mathbf{f},\ Q,\ k$)

Require: Initial queue Q
1: ▼ *the Pareto front*
2: $P \leftarrow \emptyset$
3: **while** Q.notEmpty() **do**
4: $s \leftarrow Q$.pop()
5: ▼ *recursive local improvements*
6: $s \leftarrow \mathsf{PI}(s,\ \mathbf{f})$
7: ▼ *s undominated by P*
8: **if** $\forall p \in P : \mathbf{f}(s) \not\prec \mathbf{f}(p) \wedge \mathbf{f}(s) \neq \mathbf{f}(p)$
9: $P \leftarrow \mathsf{merge}(P,\ \{s\})$
10: ▼ *new candidates*
11: $N \leftarrow \{s' \in \mathcal{N}(s) : s \not\succ s'\}$
12: Q.addK($N,\ k$)
13: **end if**
14: **end while**
15: **return** P

Algorithm 2 GQPLS($\alpha,\ p_M$)

Require: A random initial Q
1: $P \leftarrow \mathsf{QPLS}(Q,\ \mathcal{I})$
2: **while** NOT Stopping condition **do**
3: Q.clear()
4: **for** $s \in P$ **do**
5: **if** $\alpha > U(0,1)$ or $|P| < 2$
6: $s' \leftarrow \mathsf{Mutate}(s,\ p_M)$
7: **else**
8: Select $s_1 \neq s$ from P
9: $s' \leftarrow \mathsf{Recombine}(s,\ s_1)$
10: **end if**
11: Q.add(s')
12: **end for**
13: $P \leftarrow \mathsf{merge}(P,\ \mathsf{QPLS}(Q))$
14: **end while**
15: **return** P

The algorithm starts with an initial queue Q of candidate solutions, and an empty Pareto archive (line 2). Until the queue is empty, these candidate solutions are popped one by one (line 3–14). When a solution s is obtained from the queue, it first runs a recursive *Pareto improvement* function (PI) at line 6. PI improves solutions by repeatedly selecting a dominating solution from the neighborhood, until such improvements are no longer possible.

After a solution s is found that is not dominated by any of its neighbors, it is compared to the Pareto archive P (line 8), and when it is not weakly dominated by any solution in P, s is merged into P (line 9). Then, new candidate solutions are selected from the neighbors of s. The set N on line 11 is the set of neighbors of s that are incomparable to s. From N, k candidates are randomly selected to be added to the queue. By setting the parameter k, the amount of exploration in the neighborhood of one solution can be controlled. Making k too large leads to exploring a lot of unsuccessful candidates initially, whereas making k too small reduces the archive's genetic diversity. We typically choose k between 2 and 10.

Following previous work [5, 8], we can distinguish two strategies for PI, a *best-improvement* and a *first-improvement* implementation. Unlike SPLS, where improvements are applied once per iteration, we apply the improvements recursively until no improvement can be found. Because solutions that have not yet been optimized are protected in the queue, we do not have to worry about premature deletions from our intermediate Pareto archive.

QPLS guarantees that all solutions in the Pareto archive are Pareto-undominated with respect to their neighborhoods at any given time while the algorithm runs. Furthermore, it can be proved that, in a finite state space, QPLS terminates in a finite number of steps for any finite initial queue. This is because there can be

only a limited number of steps to any archive P^*, and because there can be no cycles in its execution.

However, QPLS converges to a locally optimal set. A naive method to find better approximate Pareto fronts than the single locally optimal set returned by QPLS is just to restart QPLS with different random initial queues, and *merge* the result, i.e., *multi-start QPLS* (MQPLS). However, MQPLS does not exploit the results of the individual QPLS runs to focus on more promising regions.

4 Genetic QPLS

Genetic QPLS (GQPLS) escapes local optima by mutating and recombining the entire Pareto archive that result from individual QPLS runs. After a single QPLS run, we can mutate all the solutions in the archive, and restart PLS with these mutated solutions as input. In the case of QPLS, this input is the initial queue.

The *genetic local search* (GLS) scheme combines mutation and recombination. We define GQPLS in Algorithm 2. It escapes local optima by restarting QPLS with a set of new solutions, consisting of mutations and recombinations of solutions from Pareto archive P returned by single QPLS runs. The implementation of the genetic operators Mutate and Recombine are dependent on the problem.

GQPLS starts by running QPLS on an initial queue Q to find a locally optimal Pareto archive P on line 1. On lines 3 to 12, a new queue is created which consists of mutations and recombinations from the previous archive. For each solution $s \in P$, either (with probability α) a mutation of s is generated (line 6), or (with probability $1 - \alpha$) we use a recombination of s with a random of solution $s' \neq s$ from the archive (line 9). The newly generated solution is then added to the new queue. Finally, QPLS is called again with the new queue, and the result of which is merged into P. Because we use the Pareto dominance relationship in merge, the archive can only improve or remain the same. GQPLS runs indefinitely or until some stopping condition is met.

When the crossover probability $\alpha = 0$, one could regard this as an *iterated* [4, 10] version of QPLS. We view iterated QPLS as a special case of GQPLS.

5 Experiments

We compare QPLS to existing PLS and non-PLS MOEAs on randomly generated *multi-objective coordination graphs* (MO-CoGs) [16], which are single-state problems from the multi-agent literature in which agents must work together in order to obtain a shared (vector-valued) reward. Not only do these problems form an important problem on their own (e.g., for resource gathering [16] or risk-sensitive combinatorial auctions [11]), they are also a key subproblem in more elaborate sequential settings (such as transport network maintenance planning [14]). In these settings, time is often limited, making fast heuristic methods key to their applicability. Additionally, MO-CoGs allow for fast evaluation of a mutation of an evaluated solution, which makes it possible to perform faster local search. MO-CoGs form a class of flexible and scalable problems that can vary in size (the number of variables and the size of the search space) and in complexity (the complexity of the graphs), making them interesting MOEA benchmarks.

MO-CoGs are MOOPs in which $\mathcal{V}$ is a set of n enumerated agents, $\mathcal{S} = \mathcal{A} = \mathcal{A}_1 \times \ldots \times \mathcal{A}_n$ is the Cartesian product of the action spaces of the individual agents. A specific joint action is denoted $\mathbf{a}$. Finally, $\mathbf{f}$ is a vector sum over a set of local payoff functions $\mathcal{U}$: $\mathbf{f}(\mathbf{a}) = \sum_{\mathbf{u}_e \in \mathcal{U}} \mathbf{u}_e(\mathbf{a}_e)$. A local payoff function $\mathbf{u}_e$ has limited scope e, i.e., only a subset of agents participate in it. A local joint action of the agents that participate in $\mathbf{u}_e$ is denoted $\mathbf{a}_e$. The local joint actions $\mathbf{a}_e$ on the righthand side are derived from the full joint action $\mathbf{a}$ on the lefthand side.

The local payoff functions and the agents can be seen as the two sets of nodes in a bipartite graph whose edges indicate which agents participate in which local payoff functions. A small example graph with 3 agents (circles) and 2 local payoff functions is shown in Fig. 1.

To generate MO-CoGs, we follow the random graph generation procedure proposed by Roijers et al. [16]. This procedure takes the following input variables: n, the number of agents; d, the number of objectives; ρ the number of local payoff functions; and $|\mathcal{A}_i|$, the action space size, which in this procedure is the same for all agents. The values in each local payoff function are filled with real numbers drawn independently from a uniform distribution on the interval $0 < u_{e,i}(\mathbf{a}_e) < 10$.

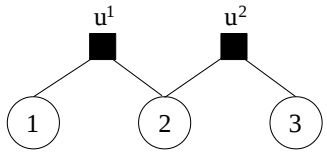

Fig. 1. A coordination graph

The payoff for a single local function depends on the actions of all connected agents, and each agent can participate in several local payoff functions. Therefore, getting good payoffs requires carefully coordinated actions. Randomly changing the action of random agents in a carefully balanced solution can therefore cause the coordination to collapse. To minimize the impact of changes in a solution but still be able to escape from local optima, the function Recombine copies the actions for adjacent agents from each solution. Specifically, it uses the graph structure to decide where it is divided: starting with a random payoff function, it adds the action from the first solution and iterates breadth-first over the agents in the graph, stopping with a probability p_S or when half of the graph has been covered. The other actions are taken from the second solution. p_S was chosen so that the expected number of actions changed is a third of the total.

The Mutate operator, by contrast, does not take graph structure into account. It returns a new solutions that is a copy of its argument, except that every action has a probability $p_M = 0.05$ to switch to a random action in its action space.

We compare against two non-PLS-employing MOEAs: NSGA-II and SPEA2, and against the PLS-employing method we emprically found best for MO-CoGs: Drugan and Thierens' first-improvement genetic SPLS [5]. We employ the same Mutate and Recombine operators described above for all methods. All algorithms, as well as the problem, were implemented in JAVA. All experiments were run on an Intel Core i7-3610QM quad core CPU at 2.30GHz.

5.1 Parameter optimization

For MO-CoGs of varying size, we optimized the parameters for each algorithm separately. For SPEA2 a low probability of crossover (0.2) was better than only using mutation. For NSGA-II 0.0 was best.

Genetic SPLS yielded the best results when the mutation probability α was set to 0.5. The first Pareto-improvement exploration strategy (as was also found by Drugan and Thierens [5] for quadratic assignment problems) was best.

Surprisingly, for Genetic QPLS the mutation only probability was best set to $\alpha = 1$. This can be explained by the observation that GQPLS has little trouble finding good solutions in the center of the Pareto front but more difficulty finding them around the edges of the Pareto front (as shown Fig. 4, which we discuss further in Sect. 5.3). Recombining values from the center of the front may be less likely to yield results on the edges, where the combination of two solutions from different edges would result in a solution around the center.

For GQPLS, recursive best Pareto-improvement performed best. This contradicts findings for earlier PLS algorithms [9], including GSPLS, the best other PLS algorithm for this problem. This can be explained by the protection of candidate solutions in the queue of QPLS. Best-improvement takes bigger steps on average; therefore, in other PLS algorithms, the probability that promising solutions in the Pareto archive are dominated by the resulting improvements and thus prematurely deleted, is higher as well. Because QPLS does not run the risk of deleting other candidate solutions from the queue, the best possible improvement strategy just speeds up the search.

The maximal numbers of additions to the queue in QPLS was best set to $k = 5$. When k was set higher, the individual runs of QPLS took longer, with only slightly better results (that were more easily achieved with the genetic scheme), and when k was lower the Pareto archives were initially narrower.

5.2 Approximation of the Pareto Front

We generated small MO-CoGs for which P^* was computed exactly using *multi-objective bucket elimination* (MO-BE) [18]. We generated 2-objective MO-CoGs of $n = 5$ to $n = 30$ agents, with $\rho = 1.5n$ and $|\mathcal{A}_i| = 10$. For larger MO-CoGs, it is not feasible to compute the true Pareto front, as MO-BE has a runtime that is exponential in the number of agents [16], and for more than 30 agents, its memory requirements become intractable.

Pareto-set approximations are evaluated in terms of the hypervolume [22] as a function of runtime. The hypervolume is defined as the volume of reward space that is dominated between the positive reward origin and the Pareto archive.

In a large portion of MOEA literature, methods are compared using the number of fitness evaluations instead of time. However, PLS methods do not do full fitness evaluations quite as often. Instead, they perform many tiny changes on a local level, which can be evaluated in a fraction of the time, and only evaluate fitness fully after recombinations and mutations. As a result, comparing the number of fitness evaluations gives an incomplete and possibly misleading picture of the computational costs. Hence, we measure hypervolume as a function of runtime rather than number of evaluation function calls.

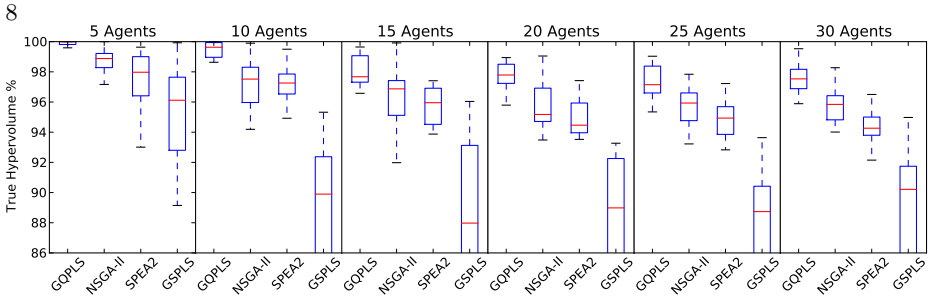

Fig. 2. Percentage of the real hypervolume found by GQPLS, NSGA-II, SPEA2 and GSPLS after 15 minute runs on MO-CoGs of varying size; 10 runs per problem size.

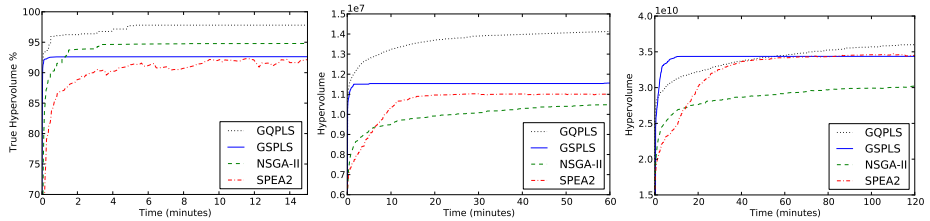

Fig. 3. The hypervolume as a function of time for one randomly generated MO-CoG for all tested MOEAs. (left) An $n = 30$, $d = 2$ MO-CoG, with hypervolume as a percentage of the hypervolume of the true Pareto front. (middle) An $n = 300$, $d = 2$ MO-CoG. (right) An $n = 300$, $d = 3$ MO-CoG.

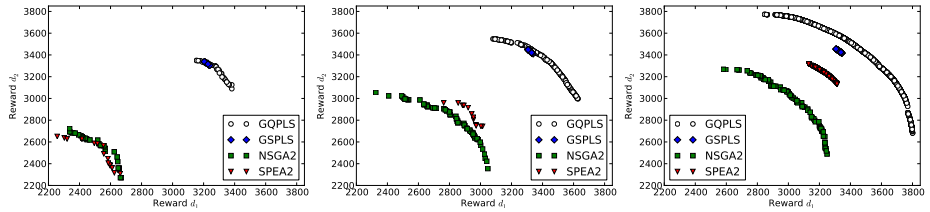

Fig. 4. The Pareto-archives found by GQPLS, GSPLS, NSGA-II and SPEA2 on a $n = 300$ MO-CoG: (left) after 20 seconds, (middle) 5 minutes, and (right) 1 hour.

Figure 2 shows the fraction of the hypervolume of P^* that is found by the MOEAs within 15 minutes in quantiles. All methods produce less accurate approximations as the problem size increases. However, GQPLS performs better in terms of best, worst, and mean results than the other methods.

Figure 3 (left) shows the growth of the hypervolume over time for an $n = 30$ MO-CoG. After 15 minutes, all methods are still improving their Pareto fronts but GQPLS has better approximations than the other algorithms at every timestep. When comparing GQPLS to MO-BE in the 30-agent setting, we found that after one iteration, which on average took 0.37% of the time required by MO-BE to calculate the true Pareto front, GQPLS found a Pareto archive that covered 92.4% of the hypervolume of P^*.

We conclude that GQPLS can approximate P^* better and more quickly than GSPLS, NSGA-II and SPEA2 for small problems, and that the difference in approximation quality tends to get bigger as the problem size increases. Furthermore, GQPLS provides good results in a fraction of the time MO-BE takes to find P^*.

5.3 A Large MO-CoG

One the strengths of MOEAs lies in the fact that they can return approximate results when calculating the exact Pareto front is intractable. Figure 3 shows the results for $n = 300$, $d = 2$ (middle) and $n = 300$, $d = 3$ (right). In the $d = 2$ problem, GQPLS outperforms the other algorithms by a large margin, while the difference is smaller in the $d = 3$ problem. A likely explanation is that the benefit of protecting diversity (by using the queue) provides a smaller benefit in higher dimensional problems, as there are more ways in which solutions can be Pareto-equivalent, and thus fewer solutions are dominated and discarded.

Figure 4 shows how the fronts develop over time. Both NSGA-II and SPEA2 have a wide spread from early on and move slowly towards a higher reward in both dimensions. Both GSPLS and GQPLS improve towards the center of the graph first, but where GSPLS has difficulty widening the front, GQPLS is able to explore the front's edges while also finding improvements in the center. We hypothesize that this is because improvements on the edges are often the result of mutations that first take a rather big step back in both objectives and only turn out well after a full recursive Pareto-improvement run. It is precisely these mutations that need to be protected from premature deletion until such improvement has taken place.

Overall, these results show that, like other PLS-based methods, GQPLS finds a relatively good approximate front quickly and continues to improve in order to find the best approximations. We therefore conclude that using a PLS method that employs queues to protect promising candidates during Pareto local search can lead to great speed-ups in MOEAs.

6 Discussion & Conclusion

In this paper, we proposed a PLS algorithm based on the principle that newly mutated solutions should be allowed to find their full potential before comparing them to other solutions. QPLS therefore protects these solutions in a queue. We embedded QPLS in a genetic search scheme, yielding Genetic QPLS, to escape from local optima. QPLS terminates and GQPLS finds the true Pareto front in the limit. We showed empirically that QPLS outperforms other popular evolutionary multi-objective algorithms on random generated multi-objective coordination graphs, where mutations of known sulutions can be evaluated efficiently.

In future work, we aim to employ ideas from other successful algorithms, such as NSGA-II and SPEA2, and focus the search on those regions of the value space that are less "crowded" by other solutions. The crowding distance, as defined by NSGA-II, can be used for such purposes. There are several ways the crowding distance could be employed: (1) embed it inside QPLS to let solutions with a higher crowding distance produce more candidate solutions in the queue, (2) use it inside recursive best Pareto improvement to discriminate between Pareto incomparable solutions, or (3) use it to seed new initial queues in the outer loop of GQPLS. Furthermore we would like to compare GQPLS to more recent MOEAs such as those reviewed by Zavala et al. [20].

Acknowledgements This research is supported by the NWO DTC-NCAP (#612.001.109) project.

References

1. C. A. C. Coello and G. B. Lamont. *Applications of multi-objective evolutionary algorithms*, volume 1. World Scientific, 2004.
2. C. A. C. Coello, G. B. Lamont, and D. A. Van Veldhuisen. *Evolutionary algorithms for solving multi-objective problems*. Springer, Heidelberg, 2007.
3. K. Deb, S. Agrawal, A. Pratap, and T. Meyarivan. A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: NSGA-II. *LNCS*, 1917:849–858, 2000.
4. M. M. Drugan and D. Thierens. Path-guided mutation for stochastic Pareto local search algorithms. In *Parallel Problem Solving from Nature, PPSN XI*, pages 485–495. Springer, Heidelberg, 2010.
5. M. M. Drugan and D. Thierens. Stochastic Pareto local search: Pareto neighbourhood exploration and perturbation strategies. *J Heur*, 18(5):727–766, 2012.
6. R. C. Eberhart and Y. Shi. Comparison between genetic algorithms and particle swarm optimization. In *Evolutionary Programming VII*, pages 611–616. Springer, Heidelberg, 1998.
7. C. Erbas, S. Cerav-Erbas, and A. D. Pimentel. Multiobjective optimization and evolutionary algorithms for the application mapping problem in multiprocessor system-on-chip design. *Evolutionary Computation, IEEE Transactions on*, 10(3):358–374, 2006.
8. P. Hansen and N. Mladenović. First vs. best improvement: An empirical study. *Discrete Appl Math*, 154(5):802–817, 2006.
9. A. Liefooghe, J. Humeau, S. Mesmoudi, L. Jourdan, and E.-G. Talbi. On dominance-based multiobjective local search: design, implementation and experimental analysis on scheduling and traveling salesman problems. *J Heur*, 18(2):317–352, 2012.
10. H. R. Lourenço, O. C. Martin, and T. Stützle. Iterated local search. *arXiv preprint math/0102188*, 2001.
11. R. Marinescu. Efficient approximation algorithms for multi-objective constraint optimization. In *ADT*, pages 150–164. Springer, Heidelberg, 2011.
12. S. Obayashi, S. Jeong, and K. Chiba. Multi-objective design exploration for aerodynamic configurations. *AIAA*, 4666:2005, 2005.
13. L. Paquete, M. Chiarandini, and T. Stützle. Pareto local optimum sets in the biobjective traveling salesman problem: An experimental study. In *Metaheuristics for Multiobjective Optimisation*, pages 177–199. Springer, Heidelberg, 2004.
14. D. M. Roijers, J. Scharpff, M. T. Spaan, F. A. Oliehoek, M. de Weerdt, and S. Whiteson. Bounded approximations for linear multi-objective planning under uncertainty. In *ICAPS*, 2014.
15. D. M. Roijers, P. Vamplew, S. Whiteson, and R. Dazeley. A survey of multi-objective sequential decision-making. *JAIR*, 47:67–113, 2013.
16. D. M. Roijers, S. Whiteson, and F. A. Oliehoek. Computing convex coverage sets for multi-objective coordination graphs. In *Algorithmic Decision Theory*, pages 309–323. Springer, Heidelberg, 2013.
17. D. M. Roijers, S. Whiteson, and F. A. Oliehoek. Linear support for multi-objective coordination graphs. In *Proceedings of the 2014 international conference on Autonomous agents and multi-agent systems*, pages 1297–1304. International Foundation for Autonomous Agents and Multiagent Systems, 2014.
18. E. Rollón and J. Larrosa. Bucket elimination for multiobjective optimization problems. *Journal of Heur*, 12(4-5):307–328, 2006.
19. P. Vamplew, R. Dazeley, A. Berry, R. Issabekov, and E. Dekker. Empirical evaluation methods for multiobjective reinforcement learning algorithms. *Mach Learn*, 84(1-2):51–80, 2011.
20. G. Zavala, A. Nebro, F. Luna, and C. A. C. Coello. A survey of multi-objective metaheuristics applied to structural optimization. *Struct Multidiscip O*, pages 1–22, 2013.
21. E. Zitzler, M. Laumanns, and L. Thiele. SPEA2: Improving the strength Pareto evolutionary algorithm, 2001.
22. E. Zitzler and L. Thiele. Multiobjective optimization using evolutionary algorithms—a comparative case study. In *Parallel problem solving from nature—PPSN V*, pages 292–301. Springer, Heidelberg, 1998.