Adaptive "Anytime" Two-Phase Local Search

Jérémie Dubois-Lacoste, Manuel López-Ibáñez, and Thomas Stützle

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

Abstract. Two-Phase Local Search (TPLS) is a general algorithmic framework for multi-objective optimization. TPLS transforms a multi-objective problem into a sequence of single-objective ones by means of weighted sum aggregations. This paper studies different sequences of weights for defining the aggregated problems for the bi-objective case. In particular, we propose two weight setting strategies that show better anytime search characteristics than the original weight setting strategy used in the TPLS algorithm.

1 Introduction

Many optimization problems require the consideration of several, conflicting objectives. In particular, multi-objective combinatorial optimization problems are currently the focus of considerable research efforts. The available approaches for tackling these problems with stochastic local search (SLS) algorithms can roughly be classified as following two main search paradigms [1]: algorithms that follow a component-wise acceptance criterion (CWAC search model), and those that follow a scalarized acceptance criterion (SAC search model). A paradigmatic example of the latter class of algorithms is two-phase local search (TPLS) [2].

TPLS starts with a high quality solution for a single objective (first phase) and then solves a sequence of scalarizations of the multi-objective problem (second phase). In the original version [2], each successive scalarization uses a slightly modified weight vector and starts from the best solution found by the previous scalarization. This approach has shown promising results for a number of multi-objective problems. In fact, TPLS is an essential component of state-of-the-art algorithms for the bi-objective traveling salesman problem [3] and the bi-objective permutation flowshop scheduling problem (PFSP) [4].

A drawback of the weight setting strategy used in TPLS is that the number of weights (or the overall available computation time) must be known in advance in order to equally distribute the computational effort among the different scalarizations. This also means that interrupting TPLS before it has explored all scalarizations is likely to produce a poor quality approximation of the Pareto front. In this sense, we may assume that TPLS has poor *anytime* properties. In fact, a central idea behind *anytime algorithms* [5] is to make a special effort to produce a result with as high quality as possible, independently of the computation time allowed. In other words, an algorithm should be designed in such a way that for each possible stopping time it reaches a performance as high as

C. Blum and R. Battiti (Eds.): LION 4, LNCS 6073, pp. 52–67, 2010.

possible. In this paper, we propose new *anytime* weight setting strategies that ensure a fair distribution of the computational effort along the nondominated front independently of a priori knowledge about the number of scalarizations.

We study two types of anytime weight setting schemes. The first one uses a regular distribution of the weight vectors. However, we noticed that its performance could further be improved by dynamically adapting the weights in dependence of the shape of the nondominated front. Our *adaptive* TPLS is inspired by the *dichotomic* method of Aneja and Nair [6] for obtaining all extreme supported non-dominated solutions in bi-objective optimization problems. We extend the dichotomic method in two significant ways. First, our adaptive approach tries to fill as early as possible the larger gaps in the nondominated front in order to satisfy the *anytime* property. Second, we extend the algorithm to use solutions found by previous scalarizations as seeds for further scalarizations, therefore, providing an adaptive TPLS strategy.

We test these strategies on the PFSP, for which we study two bi-objective problems: minimize both makespan and total flowtime, and minimize both makespan and total tardiness. The TPLS version we use in this study is the same as the one underlying the new state-of-the-art SLS algorithms for these problems [4].

2 Two-Phase and Double Two-Phase Local Search

TPLS [2] is a general algorithmic framework for multi-objective optimization that consists of two phases. The first phase uses an effective single-objective algorithm to find a high quality solution for one objective. The second phase solves a sequence of *scalarizations*, that is, weighted sum aggregations of the multiple objectives into a single scalar function. In this paper, we focus on bi-objective problems. Hence, given a normalized weight vector $\boldsymbol{\lambda} = (\lambda, 1-\lambda)$, $\lambda \in [0,1] \subset \mathbb{R}$, the scalar value of a solution s with objective function vector $\boldsymbol{f}(s) = (f_1(s), f_2(s))$ is computed as

$$f_\lambda(s) = \lambda \cdot f_1(s) + (1-\lambda) \cdot f_2(s). \tag{1}$$

The motivation for solving a scalarized problem is to exploit the effectiveness of single-objective algorithms. One central idea of TPLS is to use the best solution found by the previous scalarization as the initial solution for the next scalarization. This strategy tries to exploit the connectedness of solutions, that is, solutions that are close to each other in the solution space are expected to be close in the objective space. There are two main TPLS strategies in the literature:

Single direction (*1to2* or *2to1*). The simplest way to define a sequence of scalarizations is to use a regular sequence of weight vectors from either the first objective to the second or vice versa. We call these alternatives *1to2* or *2to1*, depending on the direction followed. For example, the successive scalarizations in *1to2* are defined by the weights $\lambda_i = 1 - ((i-1)/(N_{\text{scalar}}-1))$, $i = 1, \ldots, N_{\text{scalar}}$, where N_{scalar} is the number of scalarizations. (For simplicity, we henceforth denote weight vectors by their first component.) In *2to1* the sequence would be

Algorithm 1. Two-Phase Local Search

```
1: π1 := SLS1()
2: π2 := SLS2()
3: Add π1, π2 to Archive
4: if 1to2 then π' := π1 else π' := π2
5: for each weight λ do
6:    π' := SLSΣ(π', λ)
7:    Add π' to Archive
8: end for
9: Filter(Archive)
10: Output: Archive
```

reversed. There are two drawbacks of this strategy. First, the direction chosen gives a clear advantage to the starting objective, that is, the Pareto front approximation will be better on the starting side. Second, one needs to know in advance the computation time that is available, in order to define appropriately the number of scalarizations and the time spent on each scalarization. Algorithm 1 gives the pseudo-code of the single direction TPLS. We denote by SLS_1 and SLS_2 the SLS algorithms to minimize the first and the second single objectives, respectively. SLS_Σ is the SLS algorithm to minimize the scalarized problem. In our implementation, we first generate a very good solution for each objective because we have good algorithms for each single-objective problem, but we only use one of them as a starting seed for further scalarizations.

Double strategy. We denote as Double TPLS (D-TPLS) the strategy that first goes sequentially from one objective to the other one, as in the usual TPLS. Then, another sequence of scalarizations is generated starting from the second objective back to the first one. This is, in fact, a combination of *1to2* and *2to1*, where half of the scalarizations are defined sequentially from one objective to the other, and the other half in the other direction. This approach, proposed also by Paquete and Stützle [2], tries to avoid giving advantage to the starting objective. To introduce more variability, in our D-TPLS implementation, the weights used in the first TPLS pass are alternated with the weights used for the second TPLS pass. D-TPLS still requires to define the number of weights, and hence, the computation time, in advance.

3 The Regular "Anytime" Weight Setting Strategy

The strategy of defining successive weight vectors based on a strategy of minimal weight vector changes allows to generate very good approximations to the Pareto front of the areas "covered" by the weight vectors [2,3]. However, if stopped prematurely, it leaves other areas of the Pareto front essentially unexplored. Here, we propose a first, rather straightforward *anytime* TPLS strategy. This *regular anytime* TPLS strategy (RA-TPLS) uses a regular distribution of the weights, similar to the traditional TPLS-like strategies, but gradually intensifies

the search while ensuring a fair distribution of the computational effort along the Pareto frontier. At each iteration, a new weight is defined in the middle of the interval of two previous consecutive weights. This procedure defines a series of levels, each level providing a finer approximation to the Pareto front. The sequence of weights used by RA-TPLS is:

Level 0: $\lambda = 0, 1$ % (initial solutions)
Level 1: $\lambda = 0.5$
Level 2: $\lambda = 0.25, 0.75$
Level 3: $\lambda = 0.125, 0.375, 0.625, 0.875$
....

At each level, the number of weights and, hence, scalarizations, increases and the exploration of the Pareto front becomes successively more intense, in some sense filling the gaps in the Pareto front. Once the algorithm has completely finished a level of the search, the computational effort has been equally distributed in all directions. If the search stops before exploring all scalarizations at a certain level, the search would explore some areas of the Pareto front more thoroughly than others. In order to minimise this effect, RA-TPLS considers the weights within a level in a random order.

In RA-TPLS, following the principles of TPLS, each new scalarization (using a new weight) starts from a solution generated by a previous scalarization. In particular, RA-TPLS selects the seed of a new scalarization among the two solutions that were obtained by the previous scalarizations using the two weight vectors closer to the new weight. The algorithm computes the scalar values of these two solutions according to the new weight, and selects the solution with the best value as the seed for the new scalarization.

Algorithm 2 describes RA-TPLS in detail. There are three main data structures: L_i is a set of pairs of weights used in the previous iteration of the algorithm; S is a potential set of initial solutions together with the corresponding weight that was used to generate them; Archive is the archive of nondominated solutions. First, one initial solution is obtained for each objective using appropriate single-objective algorithms, $\mathsf{SLS}_1()$ and $\mathsf{SLS}_2()$. These new solutions and their corresponding weights, $\lambda = 1$ and $\lambda = 0$, respectively, are used to initialize L_0 and S. At each iteration, a pair of weights $(\lambda_{\text{sup}}, \lambda_{\text{inf}})$ is subtracted randomly from L_i and used to calculate a new weight given by $\lambda = (\lambda_{\text{sup}} + \lambda_{\text{inf}})/2$. Then, procedure ChooseSeed uses this weight to choose a solution from the set of seeds.

ChooseSeed finds, given a weight λ, the two solutions from the set of seeds S that have been generated using the weights closest to λ:

$$s_{\text{inf}} = \left\{ s_i \mid \max_{(s_i,\lambda_i)\in S} \{\lambda_i : \lambda_i < \lambda\} \right\} \qquad s_{\text{sup}} = \left\{ s_i \mid \min_{(s_i,\lambda_i)\in S} \{\lambda_i : \lambda_i > \lambda\} \right\} \quad (2)$$

Next, ChooseSeed calculates the scalar value of s_{sup} and s_{inf}, following Eq. (1), for the new weight λ, and returns the solution with the smaller scalar value. The chosen seed is the initial solution for SLS_Σ, the SLS algorithm used to tackle the scalarizations. The set of weights for the next iteration L_{i+1} is extended with the new pairs $(\lambda_{\text{sup}}, \lambda)$ and $(\lambda, \lambda_{\text{inf}})$. Moreover, the new solution s' and its

Algorithm 2. RA-TPLS

$s_1 := \mathsf{SLS}_1()$
$s_2 := \mathsf{SLS}_2()$
Add s_1, s_2 to Archive
$L_0 := \{(1, 0)\}$; $L_i := \emptyset \quad \forall i > 0$
$S := \{(s_1, 1), (s_2, 0)\}$
$i := 0$
while not stopping criteria met **do**
  $(\lambda_{\text{sup}}, \lambda_{\text{inf}}) :=$ extract randomly from L_i
  $L_i := L_i \setminus (\lambda_{\text{sup}}, \lambda_{\text{inf}})$
  $\lambda := (\lambda_{\text{sup}} + \lambda_{\text{inf}})/2$
  $s := \mathsf{ChooseSeed}(S, \lambda)$
  $s' := \mathsf{SLS}_\Sigma(s, \lambda)$
  Add s' to Archive
  $L_{i+1} := L_{i+1} \cup (\lambda_{\text{sup}}, \lambda) \cup (\lambda, \lambda_{\text{inf}})$
  $S := S \cup (s', \lambda)$
  $\mathsf{Filter}(S)$
  if $L_i = \emptyset$ **then** $i := i + 1$
end while
Filter(Archive)
Output: Archive

corresponding weight is added to the set of seeds, which is filtered to keep only nondominated seeds. If the current set of weights L_i is empty, the search starts using weights from level L_{i+1}. This procedure may continue indefinitely.

4 Experimental Evaluation of RA-TPLS

4.1 Bi-objective Permutation Flowshop Scheduling

As a test problem, we use bi-objective formulations of the permutation flowshop scheduling problem (PFSP), where a set of n jobs $(J_1, \ldots, J_n)$ is to be processed on m machines $(M_1, \ldots, M_m)$. Here, we study the minimization of three objectives: the makespan ($C_{\max}$), that is, the completion time of the last job; the minimization of the sum of flowtimes ($\sum C_i$), and the minimization of the total tardiness ($\sum T_i$). We considered two combinations of objectives, ($C_{\max}$,$\sum C_i$) and ($C_{\max}$,$\sum T_i$). For more details on the problems, we refer to the review paper of Minella et al. [7] and our previous work on this problem [4,8]. In fact, in our recent previous work we have developed a new, hybrid state-of-the-art SLS algorithm for these problems. A crucial components of this algorithm is a TPLS algorithm, which also forms the basis of our work here. The TPLS algorithm is based on an underlying, single-objective iterated greedy (IG) algorithm [9]. We have adapted it to the other objectives [4] and fine-tuned them using F-Race [10]. The final result was that the underlying IG algorithms reached state-of-the-art performance for all three objectives.

4.2 Experimental Setup

The two initial solutions for the weights $\lambda = 1$ and $\lambda = 0$ were obtained by running the IG algorithms for 1 000 iterations. In addition to these two solutions, we used 30 scalarizations each of 500 IG iterations. All algorithms were implemented in `C++`, and the experiments were run on a single core of Intel Xeon E5410 CPUs, running at 2.33 Ghz with 6MB of cache size, under Cluster Rocks Linux version 4.2.1/CentOS 4. For the experiments, we used 10 benchmark instances of size `50x20` and 10 instances of size `100x20` generated following the procedure described by Minella et al. [7]. Given the large discrepancies in the range of the various objectives, all objectives are dynamically normalised using the maximum and minimum values found during each run for each objective.

We examine the quality of the results in terms of the hypervolume unary measure [11,12]. In the bi-objective space, the hypervolume measures the area of the objective space weakly dominated by the solutions in a nondominated set. This area is bounded by a reference point that is worse in all objectives than all points in the nondominated front. The larger is this area, the better is a nondominated set. To compute the hypervolume, the objective values of nondominated solutions are first normalized to the range $[1, 2]$, being the values corresponding the limits of the interval slightly smaller and larger, respectively, than the minimum and the maximum values ever found for each objective. We can therefore use $(2, 2)$ as the reference point for computing the hypervolume. We present the average hypervolume as measured, for each strategy, across 15 independent runs.

4.3 Experimental Evaluation of RA-TPLS

We first study the potential benefits of the *anytime* property by comparing RA-TPLS with *1to2*, *2to1*, and D-TPLS. We compute the hypervolume value after each scalarization and give plots of the development of the hypervolume over the scalarization counter in Figure 1 on two instances of each combination of objectives (two) and instance size (two), that is, on eight instances.[1]

As expected, the hypervolume of the Pareto fronts generated by the three strategies *1to2*, *2to1*, and D-TPLS is rather poor for a small number of scalarizations, in particular, before the algorithms are allowed to reach the other objective. Interestingly, the PFSP biobjective problems also show clearly that the starting objective of TPLS can make a significant difference not only in the anytime performance but also in the final performance. Among the three, D-TPLS is clearly the one with the best final performance and also it improves the hypervolume faster than *1to2* and *2to1*. The latter indicates that for this problem, it is better to change the weights in larger steps.

By comparison, RA-TPLS quickly improves the hypervolume in very few scalarizations, and then continues to improve reaching a similar quality than D-TPLS, when the latter has performed half of its scalarizations; it is always

[1] Additional plots for all instances are available from `http://iridia.ulb.ac.be/supp/IridiaSupp2009-009`

significantly better than *1to2* and often better than *2to1*. This means that RA-TPLS outperforms the *1to2* strategy, and it produces a better result than D-TPLS and *2to1* whenever they are stopped early. However, in many instances, D-TPLS further improves the result during its second pass and, hence, we believe there is still room for improvement over RA-TPLS.

5 Adaptive Weight Setting Strategies

All weight setting strategies discussed so far generate a regular sequence of weights. That is, the weight vectors are predefined and they depend only on the number of scalarizations. In this section, we propose to dynamically generate weights in such a way that the algorithm adapts to the shape of the Pareto front. This *adaptive* strategy is inspired by the *dichotomic* scheme proposed by Aneja and Nair [6] for exact algorithms; recently, this scheme has also been adapted for the approximate case by Lust and Teghem [13]. The *dichotomic* scheme does not define the weights in advance but determines them in dependence of the solutions already found. More formally, given a pair of solutions (s_1, s_2), the new weight λ is perpendicular to the segment defined by s_1 and s_2, that is:

$$\lambda = \frac{f_2(s_1) - f_2(s_2)}{f_2(s_1) - f_2(s_2) + f_1(s_2) - f_1(s_1)} \tag{3}$$

The exact algorithm used by Aneja and Nair [6] is deterministic, and, hence, applying the same weight results in the same solution. In the scheme of Lust and Teghem [13], later scalarizations do not use as seeds the solutions found by previous scalarizations. Finally, the *dichotomic* scheme used in these two earlier papers has a natural stopping criterion, and it progresses recursively depth-first. As a result, if stopped early, it would assign an uneven computational effort along the front, possibly leading to a bad distribution of solutions and, hence, to bad anytime behavior.

We extend the *dichotomic* strategy to the TPLS framework with the goals of (i) making effective use of solutions found by previous scalarizations to seed later scalarizations; and (ii) satisfying the *anytime* property. Our resulting *adaptive* strategy is described in Algorithm 3. The main data structure is a set S of pairs of solutions found in previous scalarizations. This set is initialized with the solutions found by optimizing each single-objective using $\mathsf{SLS}_1()$ and $\mathsf{SLS}_2()$. At each iteration, the algorithm selects the pair of solutions $(s_{\text{sup}}, s_{\text{inf}}) \in S$ that define the longest segment in the objective space, using the Euclidean distance with the normalized values of each objective. The idea is to focus the search on the largest gap in the Pareto frontier in order to obtain a well-spread set of nondominated solutions. This is different from the original *dichotomic* scheme, which explores all segments recursively. Then, the algorithm calculates a new weight λ perpendicular to the segment defined by s_{sup} and s_{inf} in the objective space, following Eq. (3). Next, the underlying single-objective SLS algorithm, SLS_Σ, is run two times using the weight λ, one time starting from solution s_{sup} and one time from solution s_{inf}. This is different from the *dichotomic* strategy [6,13],

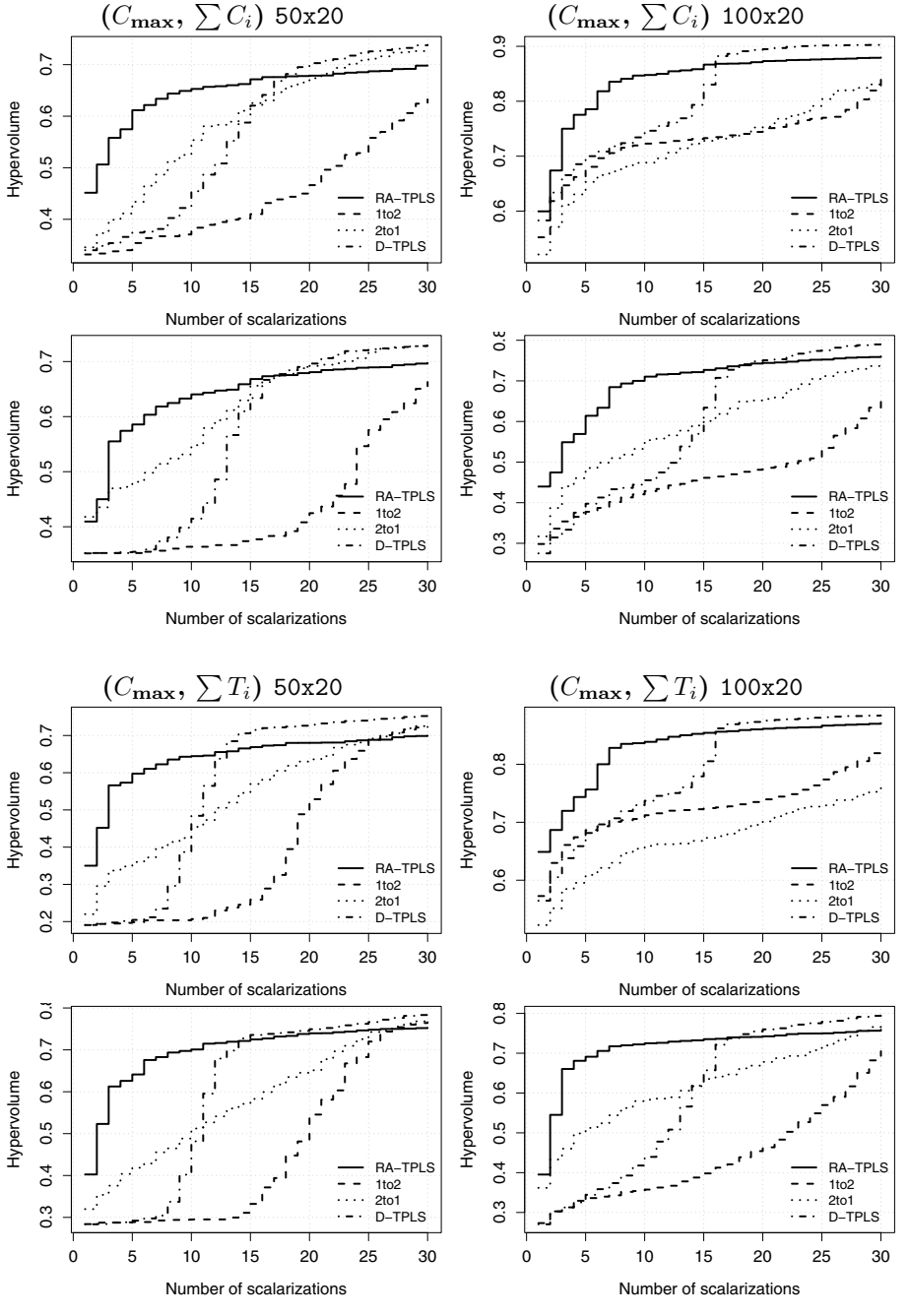

Fig. 1. Development of the hypervolume over the number of scalarizations for *1to2*, *2to1*, *D-TPLS*, and *RA-TPLS*. Results are given for four instances of size `50x20` (left column) and four instances of size `100x20` (right column). The objective combinations are $C_{\max}$, $\sum C_i$ (top four plots) and $C_{\max}$, $\sum T_i$ (bottom four plots).

Algorithm 3. Adaptive "Anytime" TPLS Strategy

1: $s_1 := \mathsf{SLS}_1()$
2: $s_2 := \mathsf{SLS}_2()$
3: Add s_1, s_2 to Archive
4: $S := \{(s_1, s_2)\}$
5: **while** not stopping criteria met **do**
6: $(s_{\text{sup}}, s_{\text{inf}}) := \arg\max_{(s,s')\in S}\{\overline{\boldsymbol{f}(s)\boldsymbol{f}(s')}\}$
7: Calculate λ perpendicular to $\overline{\boldsymbol{f}(s_{\text{sup}})\boldsymbol{f}(s_{\text{inf}})}$ following Eq. (3)
8: $s'_{\text{sup}} := \mathsf{SLS}_\Sigma(s_{\text{sup}}, \lambda)$
9: $s'_{\text{inf}} := \mathsf{SLS}_\Sigma(s_{\text{inf}}, \lambda)$
10: Add s'_{sup} and s'_{inf} to Archive
11: $\mathsf{Update}(S, s'_{\text{sup}})$
12: $\mathsf{Update}(S, s'_{\text{inf}})$
13: **end while**
14: Filter(Archive)
15: **Output:** Archive

where a scalarization is tackled only once without using as initial solution any of the previously found ones.

In the last step of an iteration, procedure Update updates the set of seeds S using the new solutions found. If s' is a new solution, any single solution in S dominated by s' is replaced with s', and any pair of solutions (weakly) dominated by s' is removed. The *dichotomic* scheme [6,13] only accepts solutions for inclusion in S if they lie *within* the triangle defined by the solutions s_{sup} and s_{inf}, and their local ideal point (see Fig. 3). Solutions outside the gray area are either dominated or not supported (not optimal for any scalarization). Heuristic algorithms may, however, generate supported solutions that are in the gray area *outside* the triangle; therefore, our *adaptive* strategy accepts *all* solutions in the gray area for inclusion in S. If a solution s' is accepted for inclusion in S, then the segment $(s_1, s_2) \in S$ with $f_1(s_1) < f_1(s') < f_1(s_2)$ is removed, and two new segments (s_1, s') and (s', s_2) are added to S. Since each iteration produces two new solutions (s'_{sup} and s'_{inf}), a maximum of three new segments are added to S every iteration. Figure 2 shows an example of the update of S after one iteration of the *adaptive* algorithm. We call this algorithm AN-TPLS in what follows (for adaptive normal TPLS).

We call *Adaptive focus* (AF-TPLS) a small variation of AN-TPLS. This variation is motivated by the fact that if two adjacent segments in S are almost parallel, two scalarizations will be solved using the same seed (the solution shared by the two segments) and very similar weights since also the two vectors perpendicular to the segments will be almost parallel. By carefully analysing the execution of AN-TPLS, we observed that such two scalarizations often generate the same solution. In order to avoid this waste of computational effort and to focus the search towards the center of each segment, we modified the calculation

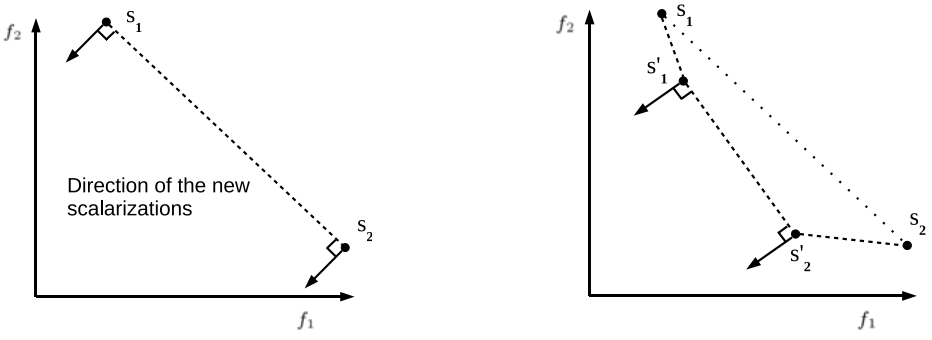

Fig. 2. A single iteration of the AN-TPLS algorithm. On the left the state before the iteration and on the right after S has been updated. The next segment which will be considered is (s'_1, s'_2) because of its larger distance.

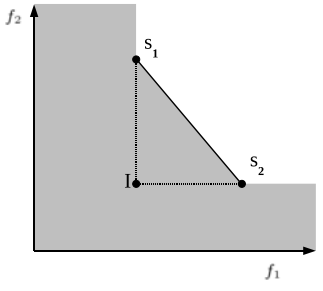

Fig. 3. Only solutions in the gray area are accepted as seeds for further scalarizations (See text for details)

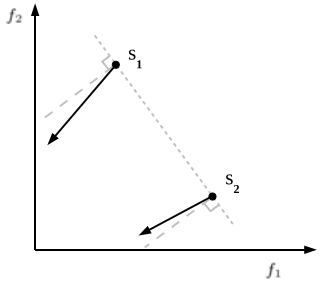

Fig. 4. AF-TPLS strategy

of the search direction of each scalarization. Given a segment $(s_1, s_2) \in S$, with $f_1(s_1) < f_1(s_2)$, we generate two weights λ_1 and λ_2 as:

$$\lambda_1 = \lambda - \theta \cdot \lambda \quad \text{and} \quad \lambda_2 = \lambda + \theta(1 - \lambda) \ . \tag{4}$$

where λ is the weight perpendicular to the segment calculated by Eq. (3), and θ is a parameter that modifies λ towards the center of the segment (see Fig. 4).

These two new weights replace the perpendicular weight λ in Algorithm 3. That is, the run of the SLS algorithm that uses s_1 as seed solves a scalarization according to weight λ_1, while the run seeded with s_2 uses the weight λ_2. A value of $\theta = 0$ would reproduce the AN-TPLS strategy.

6 Experimental Evaluation of the Adaptive TPLS Strategies

For AN-TPLS and AF-TPLS we perform the same experiments using the same experimental setup as described in Section 4; for AF-TPLS we use a setting

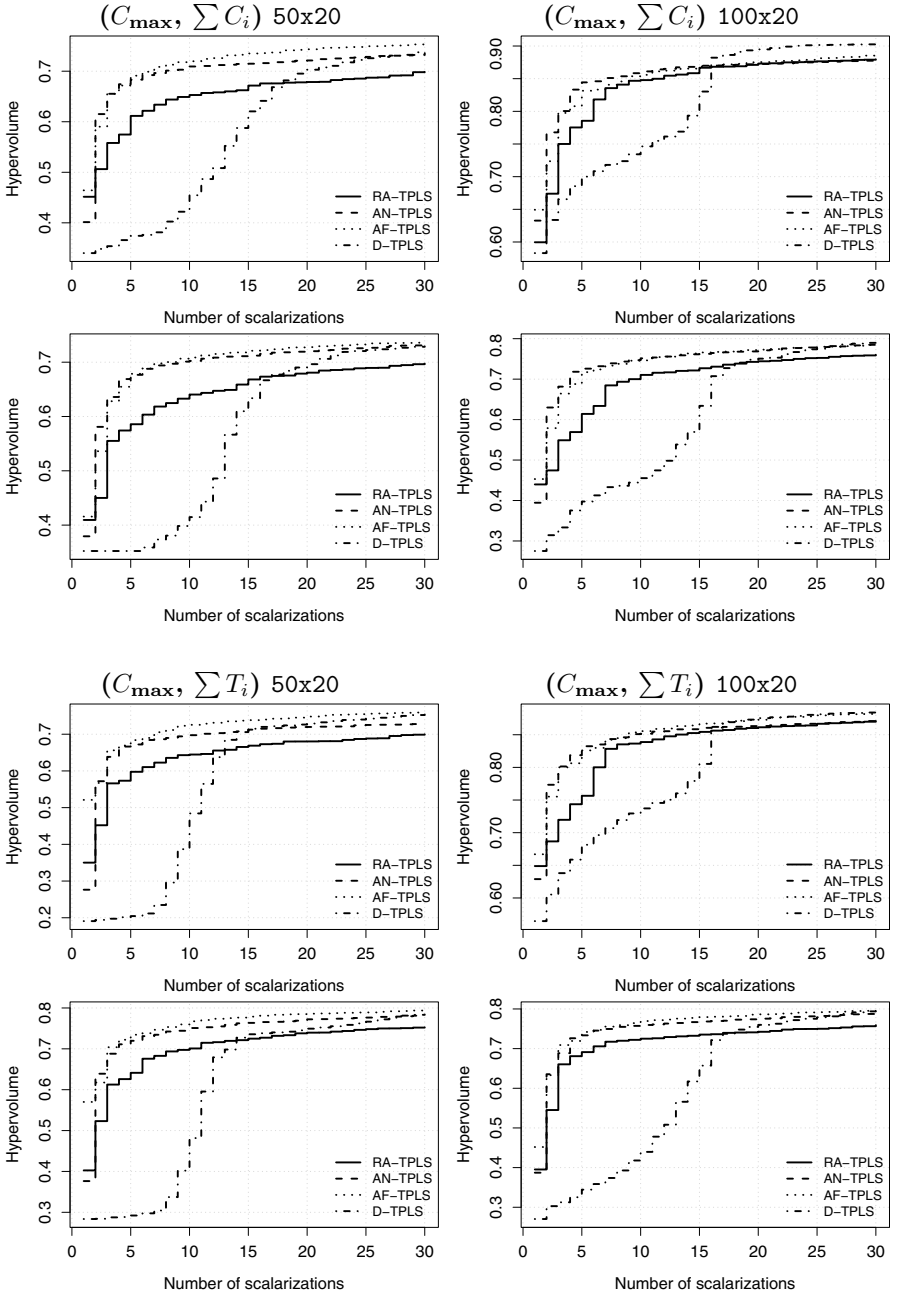

Fig. 5. Development of the hypervolume over the number of scalarizations for *D-TPLS*, RA-TPLS, AN-TPLS, AF-TPLS. Results are given for four instances of size `50x20` (left column) and four instances of size `100x20` (right column). The objective combinations are $C_{\max}$, $\sum C_i$ (top four plots) and $C_{\max}$, $\sum T_i$ (bottom four plots).

of $\theta = 0.25$. Figure 5 shows that the adaptive strategies greatly improve over the RA-TPLS strategy from the very start of the algorithm, thereby further enhancing the *anytime* behavior. In addition, the final quality is also greatly improved, reaching the hypervolume value obtained by D-TPLS. This means that the *adaptive* TPLS strategies can effectively replace D-TPLS, the former providing results similar to the latter if both are run for sufficiently long time, and much better results if the algorithms are stopped after few scalarizations. With respect to the value of θ in the *adaptive* strategy, there is not a completely clear conclusion. The *adaptive* strategy with $\theta = 0.25$ (AF-TPLS) is clearly better than AN-TPLS ($\theta = 0$) on few instances and never clearly worse. However, for most instances, and especially for the larger ones, the difference is small.

6.1 Statistical Analysis

We perform a statistical analysis of the approaches. The analysis is based on the Friedman test for analysing non-parametric unreplicated complete block designs, and its associated post-test for multiple comparisons [14]. We perform the following procedure separately for each combination of objectives, each instance size `50x20` and `100x20`, and stopping criteria of 10, 20 and 30 scalarizations. First, we calculate the median hypervolume of the 15 runs of each algorithm for each instance. Then, we perform a Friedman test using the ten instances as the blocking factor, and the different strategies as the treatment factor. In all cases, the Friedman test rejects the null hypothesis with a p-value much lower than 0.05. Hence, we proceed to calculate the minimum difference between the sum of ranks of two strategies that is statistically significant (ΔR_α), given a significance level of $\alpha = 0.05$. We examine which strategies are not significantly different to the one with the lowest sum of ranks. Table 1 summarises the

Table 1. For each number of scalarizations, strategies are ordered according to the rank obtained. The numbers in parenthesis are the difference of ranks relative to the best strategy. Strategies which are not significantly different to the best one are indicated in bold face. See text for details.

N_{scalar}	ΔR_α	**Strategies** (ΔR)
($C_{\max}$, $\sum C_i$) `50x20`		
10	2.94	**AN-TPLS (0), AF-TPLS (2),** RA-TPLS (16), 2to1(26), D-TPLS (36), 1to2(46)
20	5.23	**AN-TPLS (0), AF-TPLS (0),** D-TPLS (12), RA-TPLS (28), 2to1(30), 1to2(44)
30	6.49	**AN-TPLS (0), AF-TPLS (1), D-TPLS (3),** 2to1(20), RA-TPLS (31), 1to2(41)
($C_{\max}$, $\sum C_i$) `100x20`		
10	5.20	**AF-TPLS (0), AN-TPLS (4),** RA-TPLS (14), 2to1(28), D-TPLS (35), 1to2(46)
20	6.76	**AF-TPLS (0),** AN-TPLS (8), D-TPLS (15), RA-TPLS (21), 2to1(36), 1to2(46)
30	8.40	**D-TPLS (0), AF-TPLS (7),** AN-TPLS (12), 2to1(27), RA-TPLS (30), 1to2(44)
($C_{\max}$, $\sum T_i$) `50x20`		
10	3.79	**AF-TPLS (0),** AN-TPLS (6), RA-TPLS (18), 2to1(32), D-TPLS (34), 1to2(48)
20	3.12	**AF-TPLS (0),** AN-TPLS (10), D-TPLS (17), RA-TPLS (29), 2to1(39), 1to2(49)
30	7.10	**D-TPLS (0), AF-TPLS (1),** AN-TPLS (15), 2to1(21), 1to2(37) , RA-TPLS (40)
($C_{\max}$, $\sum T_i$) `100x20`		
10	3.60	**AF-TPLS (0),** AN-TPLS (8), RA-TPLS (19), 2to1(31), D-TPLS (38), 1to2(48)
20	7.50	**AF-TPLS (0),** AN-TPLS (11), D-TPLS (13), RA-TPLS (20), 2to1(37), 1to2(45)
30	6.43	**D-TPLS (0), AF-TPLS (3),** AN-TPLS (20), RA-TPLS (24), 2to1(34), 1to2(45)

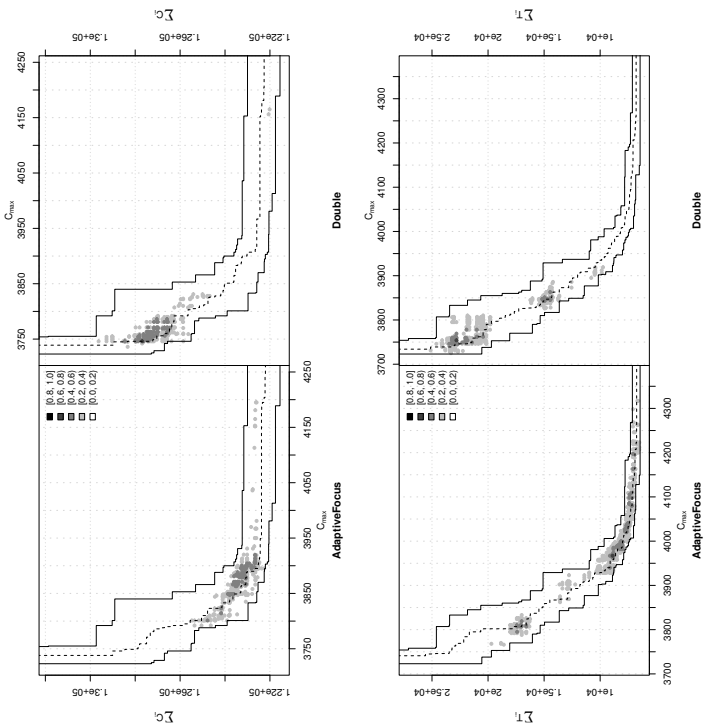

Fig. 7. EAF differences between AF-TPLS vs. D-TPLS

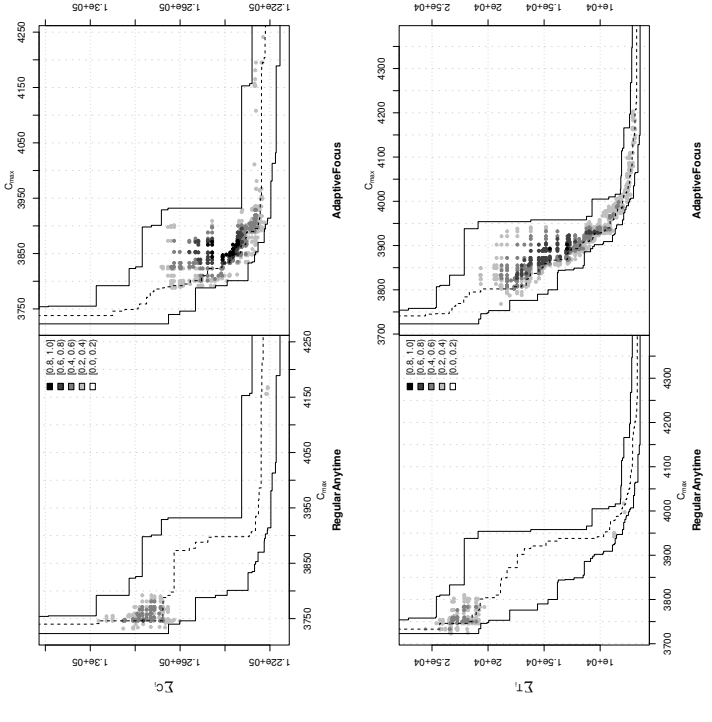

Fig. 6. EAF differences between RA-TPLS vs. AF-TPLS

results of the statistical analysis. It shows the value of ΔR_α for $\alpha = 0.05$, the strategies sorted by increasing sum of ranks; and the difference between the sum of ranks of each strategy and the best strategy (ΔR). The strategies that are not significantly different from the best are shown in boldface. The table shows that AF-TPLS is often the best, and never significantly different from the best. For a low number of scalarizations, the *adaptive* strategies (AN-TPLS and AF-TPLS) are always superior to the classical TPLS strategies. Moreover, AF-TPLS is never significantly worse than D-TPLS, when the latter runs until completion (30 scalarizations). In conclusion, AF-TPLS would be the strategy to be chosen.

6.2 Detailed Examination Based on the EAF Differences

We further explore the differences between RA-TPLS, AF-TPLS and D-TPLS by examining the empirical attainment functions (EAFs) of the final results after 30 scalarizations. The EAF of an algorithm provides the probability, estimated from several runs, of an arbitrary point in the objective space being attained by (dominated by or equal to) a solution obtained by a single run of the algorithm [15]. Examining the differences between the EAFs of two algorithms allows us to identify regions of the objective space where one algorithm performs better than another. Given a pair of algorithms, the differences in favor of each algorithm are plotted side-by-side and the magnitude of the difference is encoded in gray levels. For more details, we refer to López-Ibáñez et al. [16].

Figure 6 illustrates the EAF differences between RA-TPLS versus AF-TPLS in two instances of size `50x20`, one from each of the two combinations of objectives. It shows strong differences in favour of AF-TPLS along the whole Pareto front. Nonetheless, RA-TPLS is able to obtain very good solutions in the region minimising the makespan ($C_{\max}$). This was a consistent result among most of the instances tested. It indicates that RA-TPLS focuses excessively in a small region of the Pareto front, whereas AF-TPLS distributes its effort better along the Pareto front. By comparison, the EAF differences between AF-TPLS and D-TPLS on the same two instances (Fig. 7) show smaller (according to the gray level) and more localized differences. Therefore, no strategy can be said to outperform the other along the whole Pareto front.

The extension of the analysis above to all instances tested, let us conclude that the adaptive variants are better than RA-TPLS along the whole front. We also conclude that AN-TPLS and AF-TPLS obtain good results in different regions of the nondominated front, and the particular regions may depend on the shape of the Pareto front. Finally, D-TPLS and AF-TPLS obtain notably similar results.

7 Conclusion

In this paper, we address a perceived weakness of the TPLS framework, namely, that the number of scalarizations must be specified in advance and that stopping the algorithm earlier results in poor performance. We propose weight setting strategies that try to fulfill the *anytime* property, that is, they can be stopped

at any time during its execution, and the result would be a well-spread approximation of the Pareto front. Our first proposal, the RA-TPLS strategy, has the *anytime* property and outperforms the classical TPLS strategies if the algorithms are stopped before completion. However, its final quality is not as good as that of D-TPLS. Our second proposal is an adaptive strategy that has the *anytime* property, and it can be further fine-tuned through a parameter θ. The two adaptive variants studied here, AN-TPLS and AF-TPLS ($\theta = 0.25$), outperform the classical TPLS strategies at any time during their execution. In fact, the adaptive strategies proposed here should replace the classical TPLS strategies in situations where the computation time available is not known in advance.

Acknowledgments. This work was supported by the META-X project, an *Action de Recherche Concertée* funded by the Scientific Research Directorate of the French Community of Belgium. Thomas Stützle acknowledges support from the Belgian F.R.S.-FNRS, of which he is a Research Associate. The authors also acknowledge support from the FRFC project "*Méthodes de recherche hybrides pour la résolution de problèmes complexes*".

References

1. Paquete, L., Stützle, T.: Stochastic local search algorithms for multiobjective combinatorial optimization: A review. In: Gonzalez, T.F. (ed.) Handbook of Approximation Algorithms and Metaheuristics, pp. 29-1–29-15. Chapman & Hall/CRC (2007)
2. Paquete, L., Stützle, T.: A two-phase local search for the biobjective traveling salesman problem. In: Fonseca, C.M., Fleming, P.J., Zitzler, E., Deb, K., Thiele, L. (eds.) EMO 2003. LNCS, vol. 2632, pp. 479–493. Springer, Heidelberg (2003)
3. Paquete, L., Stützle, T.: Analysis of components of stochastic local search algorithms for the multiobjective traveling salesman problem and the design of algorithms. Computers & Operations Research 36(9), 2619–2631 (2009)
4. Dubois-Lacoste, J., López-Ibáñez, M., Stützle, T.: Effective hybrid stochastic local search algorithms for biobjective permutation flowshop scheduling. In: Sampels, M. (ed.) HM 2009. LNCS, vol. 5818, pp. 100–114. Springer, Heidelberg (2009)
5. Zilberstein, S.: Using anytime algorithms in intelligent systems. AI Magazine 17(3), 73–83 (1996)
6. Aneja, Y.P., Nair, K.P.K.: Bicriteria transportation problem. Management Science 25(1), 73–78 (1979)
7. Minella, G., Ruiz, R., Ciavotta, M.: A review and evaluation of multiobjective algorithms for the flowshop scheduling problem. INFORMS Journal on Computing 20(3), 451–471 (2008)
8. Dubois-Lacoste, J.: A study of Pareto and two-phase local search algorithms for biobjective permutation flowshop scheduling. Master's thesis, IRIDIA, Université Libre de Bruxelles, Brussels, Belgium (2009)
9. Ruiz, R., Stützle, T.: A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. European Journal of Operational Research 177(3), 2033–2049 (2007)

10. Balaprakash, P., Birattari, M., Stützle, T.: Improvement strategies for the F-Race algorithm: Sampling design and iterative refinement. In: Bartz-Beielstein, T., Blesa Aguilera, M.J., Blum, C., Naujoks, B., Roli, A., Rudolph, G., Sampels, M. (eds.) HCI/ICCV 2007. LNCS, vol. 4771, pp. 108–122. Springer, Heidelberg (2007)
11. 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)
12. Fonseca, C.M., Paquete, L., López-Ibáñez, M.: An improved dimension-sweep algorithm for the hypervolume indicator. In: IEEE Congress on Evolutionary Computation, July 2006, pp. 1157–1163. IEEE Press, Los Alamitos (2006)
13. Lust, T., Teghem, J.: Two-phase Pareto local search for the biobjective traveling salesman problem. Journal of Heuristics (2009) (to appear)
14. Conover, W.J.: Practical Nonparametric Statistics, 3rd edn. John Wiley & Sons, New York (1999)
15. Grunert da Fonseca, V., Fonseca, C.M., Hall, A.O.: Inferential performance assessment of stochastic optimisers and the attainment function. In: Zitzler, E., Deb, K., Thiele, L., Coello Coello, C.A., Corne, D.W. (eds.) EMO 2001. LNCS, vol. 1993, pp. 213–225. Springer, Heidelberg (2001)
16. 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, Springer, Heidelberg (to appear 2010)