On Universal Search Strategies for Multi-Criteria Optimization Using Weighted Sums

Julien Legriel, Scott Cotton, Oded Maler

Verimag Research Report n^{o} TR-2011-7

March 2011

On Universal Search Strategies for Multi-Criteria Optimization Using Weighted Sums

Julien Legriel, Scott Cotton, Oded Maler

March 2011

Abstract

We develop a stochastic local search algorithm for finding Pareto points for multicriteria optimization problems. The algorithm alternates between different single-criterium optimization problems characterized by weight vectors. The policy for switching between different weights is an adaptation of the universal restart strategy defined by [LSZ93] in the context of Las Vegas algorithms. We demonstrate the effectiveness of our algorithm on multicriteria *quadratic assignment problem* benchmarks and prove some of its theoretical properties.

Keywords: multi-criteria optimization, local search, restart, weighted sum, quadratic assignment problem

Reviewers:

How to cite this report:

```
@techreport {TR-2011-7,
    title = {On Universal Search Strategies for Multi-Criteria Optimization Using Weighted
Sums},
   author = {Julien Legriel, Scott Cotton, Oded Maler},
   institution = {{Verimag} Research Report},
   number = {TR-2011-7},
   year = {}
}
```

1 Introduction

The design of complex systems involves numerous *optimization* problems, where design choices are encoded as valuations of decision variables and the relative merits of each choice are expressed via a utility/cost function over these variables. In many real-life situations one has to deal with cost functions which are *multi-dimensional*. For example, a cellular phone that we want to purchase or develop can be evaluated according to its cost, screen size, power consumption and performance. A configuration s which dominates s' according to one criterium, can be worse according to another. In the absence of a linear ordering of the alternatives, there is no *unique* optimal solution but rather a set of *efficient* solutions, also known as Pareto solutions [Par12]. Such solutions are characterized by the property that their cost cannot be improved in one dimension without being worsened in another. The set of all Pareto solutions, the *Pareto front*, represents the problem trade-offs, and being able to sample this set in a representative manner is a very useful aid in decision making.

In this paper we study the adaptation of *stochastic local search* (SLS) algorithms, used extensively in (single-objective) optimization and constraint satisfaction problems, to the multi-objective setting. SLS algorithms perform a guided probabilistic exploration of the decision space where at each time instance, a successor is selected among the neighbors of a given point, with higher probability for locally-optimal points. SLS algorithms can be occasionally restarted, where restart mean abandoning the current point and starting from scratch. It has been observed (and proved) [LSZ93] that in the absence of a priori knowledge about the cost landscape, scheduling such restarts according to a specific pattern boosts the performance of such algorithm in terms of expected time to find a solution. This has led to successful algorithms for several problems, including SAT [PD07].

One popular approach for handling multi-objective problems is based on optimizing a one-dimensional cost function defined as a *weighted sum* of the individual costs according to a weight vector λ. Repeating the process with different values of λ leads to a good approximation of the Pareto front. Our major contribution is an algorithmic scheme for distributing the optimization effort among different values of λ. To this end we adapt the ideas of [LSZ93] to the multi-objective setting and obtain an algorithm which is very efficient in practice.

The rest of this paper is organized as follows. In Section 2, we introduce an abstract view of randomized optimizers and recall the problematics of multicriteria optimization. Section 3 discusses the role of restarts in randomized optimization. Section 4 presents our algorithm and proves its properties. Section 5 provides experimental results and Section 6 concludes.

2 Background

2.1 SLS Optimization

Randomization has long been a useful tool in optimization, taking quite a variety of forms, amongst which we find such different ideas as [Mat65, KGV83, AC91]. In this paper, we are concerned with optimization over finite domain functions by means of *stochastic local search* (SLS), in the style of such tools as WalkSAT [SKC93], UBCSAT [TH05] or LKH [Hel09], and well-documented in [HS04]. Here we provide a simplified overview in which we consider only a minimal characterization of SLS necessary for the paper as a whole.

We assume a finite set X called the *decision space* whose elements are called points. We assume some metric ρ on X which in a combinatorial setting corresponds to a kind of Hamming distance where $\rho(s, s')$ characterizes the number of modifications needed to transform s to s'. The neighborhood of s is the set $\mathcal{N}(s) = \{s' : \rho(s, s') = 1\}$. A cost (objective) function $f : X \rightarrow \mathbb{R}$ is defined over X and is the subject of the optimization effort. We call the range of f the *cost space* and say that a cost o is *feasible* if there is $s \in X$ such that $f(s) = o$. A random walk is a process which explores X starting from an initial randomly-chosen point s by repeatedly stepping from s to a neighboring point s', computing $f(s')$ and storing the *best* point visited so far. In each step, the next point s' is selected according to some probability distribution D_s over X which we assume to be specific to point s. Typically, D_s gives non-zero probability to a small portion of X, included in $\mathcal{N}(s)$.

While the particular probability distributions vary a great deal such processes, including [SKC93, TH05, Hel09], also share many properties. Some such properties are crucial to performance of SLS optimization algorithms. For example, all processes we are aware of favor local optima in the probability distributions from which next states are selected. The efficiency of such processes depends on structural features of the the decision space such as the size of a typical neighborhood and the maximal distance between any pair of points, as well as on the cost landscape. In particular, cost landscapes admitting vast quantities of local optima and very few global optima and where the distances between local and global optima are high, tend to be difficult for random walk algorithms [HS04, Ch. 5].

Of particular interest to this paper, a simple and common practice exploited by SLS based optimization tools is *restarting*. In our simplified model of random walk processes, a restart simply chooses a next state as a sample from a constant *initial* probability distribution rather than the next-state probability distribution. Typically the process may start over from a randomly generated solution or from a constant one which was computed off-line. Restarting can be an effective means of combating problematics associated with the cost landscape. It is particularly efficient to avoid stagnation of the algorithm and escape a region with local optimas. Most search methods feature a mechanism to thwart this behavior, but this might be insufficient sometimes. For instance a tabu search process may be trapped inside a long cycle that it fails to detect. Another aspect in which restarts may help is to limit the influence of the random generator. Random choices partly determine the quality of the output and starting from scratch is a *fast* way to cancel bad random decisions that were early made. As we will argue, restarting can also be effectively exploited for multi-criteria problems.

2.2 Multicriteria Optimization

Multi-criteria optimization problems seek to optimize a multi-valued objective $f : X \to \mathbb{R}^d$ which we can think of as a vector of costs $(f^1, f^2, \ldots, f^d)$. When $d > 1$ the cost space is not linearly-ordered and there is typically no single optimal point, but rather a set of *efficient* points known as the *Pareto front*, consisting of *non-dominated* feasible costs. We recall some basic vocabulary related to multi-objective optimization. The reader is referred to [Ehr05, Deb01] for a general introduction to the topic.

Definition 1 (Domination) *Let o and o' be points in the cost space. We say that o* dominates *o', written $o \sqsubset o'$, if for every $i \in [1..d]$, $o_i \leq o'_i$ and $o \neq o'$.*

Definition 2 (Pareto Front) *The Pareto front associated with a multi-objective optimization problem is the set O^* consisting of all feasible costs which are not dominated by other feasible costs.*

A popular approach to tackle multi-objective optimization problems is to reduce them to several single-objective ones.

Definition 3 (λ-Aggregation) *Let $f = (f^1, \ldots, f^d)$ be a d-dimensional function and let $\lambda = (\lambda^1, \lambda^2, \ldots \lambda^d)$ be a vector such that*

1. $\forall j \in [1..d], \lambda^j > 0$
2. $\sum_{j=1}^{d} \lambda^j = 1$*; and*

The λ-aggregation of f is the function $f_\lambda = \displaystyle\sum_{j=1}^{d} \lambda^j f^j$.

Intuitively, the components of λ represent the relative importance (weight) one associates with each objective. A point which is optimal with respect to f_λ is also a Pareto point for the original cost function. Conversely, when the Pareto front is convex in the cost space, every Pareto solution corresponds to an optimal point for some f^λ. At this point of discourse it is also important to explain why we choose to optimize weighted sums of the objectives when we are aware of one major drawback of this technique : the impossibility to reach Pareto points on concave parts of the front. Actually negative results [Geo67, Ehr05] only state that some solutions are not *optimum* under any combination of the objectives. This does not mean that they are unreachable with a local search algorithm because we may encounter them while making steps in the decision space. As our algorithm utilizes a Pareto filter and keeps track of the non dominated set of *all* the points it came accross, it is theoretically able to find any Pareto point if random choices are allowed. Given the graph induced on $\mathcal{X}$ by the neighborhood relation, this amounts to say that any node in the graph has a non zero probability of being visited. This fact was also confirmed experimentally (see section 5.1), as we found the whole Pareto front (including non-supported solutions) for small instances where it is known.

2.3 Related Work

Stochastic local search methods are used extensively for solving hard combinatorial optimization problems for which exact methods do not scale [HS04]. Among the well known techniques we find *simulated annealing* [KGV83], *tabu search* [GM06], *ant colony optimization* [AC91] and *evolutionary algorithms* [Mit98, Deb01] each of which has been applied to hard combinatorial optimization problems such as the traveling salesman, scheduling or assignment problems.

Many state-of-the-art local search algorithms have their multi-objective version [PS06]. For instance, there exists multi-objective extensions of simulated annealing [CJ98, BSMD08] and tabu search [GMF97]. A typical way of treating several objectives in that context is to optimize a predefined or dynamically updated series of linear combinations of the cost functions. A possible option is to pick a representative set of weight vectors a priori and run the algorithm for each scalarization successively. This has been done in [UTFT99] using simulated annealing as backbone. More sophisticated methods have also emerged where the runs are made in parallel and the weight vector is adjusted during the search [CJ98, Han97] in order to improve the diversity in the population. Weight vectors are thus modified such that the different runs are guided towards distinct unexplored parts of the cost space.

One of the main issue faced with the weighted sum method is therefore to share the time between different promising search directions (weight vectors) appropriately. Generally deterministic strategies are used: weight vectors are predetermined, and they may be modified dynamically according to a deterministic heuristic. However, unless some knowledge on the cost space has been previously acquired, one may only speculate on the directions which are worth exploring. This is why we study in this work a completely stochastic approach, starting from the assumption that every search direction has equal potential in improving the final result. The goal we seek is therefore to come up with a scheme ensuring a *fair* time sharing between different weight vectors.

Also related to this work are population-based genetic algorithms which do naturally handle several objectives as they work on several individual solutions that are mutated and recombined. They are vastly used due to their wide applicability and good performance and at the same time they also benefit from combination with specialized local search algorithms. Indeed some of the top performing algorithms for solving combinatorial problems are *hybrid*, in the sense that they mix evolutionary principles with local search. This led to the class of so called memetic algorithms [KC05]. It is common that scalarizations of the objectives are used inside them for guiding the search, and choosing a good scheme to adapt weight vectors dynamically is also an interesting issue.

The work presented in this paper began from the will to combine two ingredients used for solving combinatorial problems : scalarization of the objectives which we just discussed and restarts. Restarts have been used extensively in stochastic local search since they usually bring a non negligible improvement in the results. For instance Greedy Randomized Adaptive Search Procedures (GRASP) [FR95] or iterated local search [LMS03] are well-known methods which run successive local search processes starting from different initial solutions. In [HS04, Ch. 4] restarts in the context of single-criteria SLS are studied. Their technique is based on an empirical evaluation of run-time distributions for some classes of problems. In this work, we devise multi-criteria restarting strategies which perform reasonably well no matter what the problem or structure of the cost space happens to be. The main novelty is a formalization of the notion of a universal multicriteria restart strategy: every restart is coupled with a change in the direction and restarts are scheduled in such a way that work is balanced across directions and run times. This is achieved by adapting the ideas of [LSZ93] (which we present in the next section) to the multiobjective optimization setting.

3 Restarting Single Criteria SLS Optimizers

This practice of restarting in SLS optimization has been in use at least since [SKC93]. At the same time, there is a theory of restarts formulated in [LSZ93] which applies to *Las Vegas* algorithms which are defined for *decision problems* rather than optimization. Such algorithms are characterized by the fact that their run-time until giving a correct answer to the decision problem is a random variable. In the following we recall the results of [LSZ93] and analyze their applicability to optimization using SLS. We begin by defining properly what a strategy for restarting is.

Definition 4 (Restart Strategy) *A restart strategy S is an infinite sequence of positive integers $t_1, t_2, t_3, \ldots$. The t time prefix of a restart strategy S, denoted $S[t]$ is the maximal sequence $t_1, t_2, \ldots, t_k$ such that $\Sigma_{i=1}^{k} t_i \leq t$.*

Running a Las Vegas algorithm according to strategy S means running it for t_1 units of time, restarting it and running it for t_2 units of time and so on.

3.1 The Luby Strategy

A strategy is called *constantly repeating* if it takes the form $c, c, c, c, \ldots$ for some positive integer c. As shown in [LSZ93] every Las Vegas algorithm admits an optimal restart strategy which is constantly repeating (the case of infinite c is interpreted as no restarting). However, this fact is not of much use because typically one has no clue for finding the right c. For these reasons, [LSZ93] introduced the idea of *universal* strategies that "efficiently simulate" every constantly repeating strategy. To get the intuition for this notion of simulation and its efficiency consider first the periodic strategy $S = c, c', c, c', c, c', \ldots$ with $c < c'$. Since resets are considered independent, following this strategy for $m(c + c')$ steps amounts to spending mc time according to the constant strategy c and mc' time according to strategy c'.[1] Putting it the other way round, we can say that in order to achieve (expected) performance as good as running strategy c' for t time, it is sufficient to run S for time $t + c(t/c')$. The function $f(t) = t + c(t/c')$ is the *delay* associated with the simulation of c' by S.[2] It is natural to assume that a strategy which simulates numerous other strategies (i.e has sub-sequences that fit each of the simulated strategies) admits some positive delay.

Definition 5 (Delay Function) *A monotonic non-decreasing function $\delta : \mathbb{N} \to \mathbb{N}$, satisfying $\delta(x) \geq x$ is called a delay function. We say that S simulates S' with delay bounded by δ if running S' for t time is not better than running S for $\delta(t)$ time.*

It turns out that a fairly simple strategy, which has become known as the *Luby strategy*, is universally efficient.

Definition 6 (The Luby Strategy) *The Luby Strategy is the sequence*

$$c_1, c_2, c_3, \ldots$$

where

$$c_i \doteq \begin{cases} 2^{k-1} & \text{if } i = 2^k - 1 \\ t_{i-2^{k-1}+1} & \text{if } 2^{k-1} \leq i < 2^k - 1 \end{cases}$$

which gives

$$1, 1, 2, 1, 1, 2, 4, 1, 1, 2, 1, 1, 2, 4, 8, \ldots$$

We denote this strategy by $\mathcal{L}$.

[1] In fact, since running an SLS process for c' is at least as good as running it for c time, running S for $m(c+c')$ is at least as good as running c for $m(c+c)$ time.

[2] For simplicity here we consider the definition of the delay only at time instants $t = kc'$, $k \in \mathbb{N}$. In the case where $t = kc' + x$, where x is an integer such that $0 < x < c'$, the delay function would be $\delta(t) = \lfloor t/c' \rfloor c + t$.

A key property of this strategy is that it naturally multiplexes different constant strategies. For instance the prefix of definition 6 contains eight ones, four twos, two fours and one eight, and the total time dedicated to each strategy 1,2,4 and 8 is the same. More formally, let A denote a Las Vegas algorithm and $A(c)$ denote the algorithm which runs A for c time units. Then, the sum of the execution times spent on $A(2i)$ is equal for all $1 \leq i < 2^{k-1}$ over a $2^k - 1$ length prefix of the series. As a result, every constant restart strategy where the constant is a power of 2 is given an equal time budget. This can also be phrased in terms of delay.

Proposition 1 (Delay of $\mathcal{L}$) *Strategy $\mathcal{L}$ simulates any power of two constant strategy c with delay $\delta(t) \leq t(\lfloor \log t \rfloor + 1)$.*

Proof 1 (Sketch) *Consider a constant strategy $c = 2^a$ and a time $t = kc$, $k \in \mathbb{N}$. At the moment where the k^{th} value of c appears in $\mathcal{L}$, the previous ones in the sequence are all of the form 2^i for some $i \in \{0..\lfloor \log t \rfloor\}$. This is because the series is built such that time is doubled before any new power of two is introduced. Furthermore after the execution of the k^{th} c, every 2^i constant with $i \leq a$ has been run for exactly t time, and every 2^i constant with $i > a$ (if it exists in the prefix) has been executed less than t time. This leads to $\delta(t) \leq t(\lfloor \log t \rfloor + 1)$.*

[Sketch]

This property implies that restarting according to $\mathcal{L}$ incurs a logarithmic delay over using the optimum constant restart strategy (that we may not know) of a particular problem instance. Additionally the strategy is optimal in the sense that it is not possible to have better than logarithmic delay if we seek to design a strategy simulating *all* constant restart strategies [LSZ93]. The next section investigates how these fundamental results can be useful in the context of optimization using stochastic local search.

3.2 SLS optimizers

An SLS optimizer is an algorithm whose run-time distribution for finding a particular cost o is a random-variable.

Definition 7 (Expected Time) *Given an SLS process and a cost o, the random variable Θ_o indicates the time until the algorithm outputs a value at least as good as o.*

There are some informal reasons suggesting that using strategy $\mathcal{L}$ in the context of optimization would boost the performance. First a straightforward extension of results of 3.1 to SLS optimizers can be made.

Corollary 1 *Restarting an SLS process according to strategy $\mathcal{L}$ gives minimum expected run-time to reach any cost o for which Θ_o is unknown.*

The corollary follows directly because the program that runs the SLS process and stops when the cost is at least as good as o is a Las Vegas algorithm whose run-time distribution is Θ_o. In particular using strategy $\mathcal{L}$ in that context gives a *minimal expected time to reach the optimum* o^*. Still, finding the optimum is too ambitious for many problems and in general we would like to run the process for a fixed time t and obtain the best approximation possible within that time. A priori there is no reason for which minimizing the expected time to reach the optimum would also maximize the approximation quality at a particular time t.

On the contrary for each value of o we have more or less chances to find a cost at least as good as o within time t depending on the probability $P(\Theta_o \leq t)$.[3] Knowing the distributions one could decide which o is more likely to be found before t and invest more time for the associated optimal constant restart strategies. But without that knowledge every constant restart strategy might be useful for converging to a good approximation of o^*. Therefore whereas Las Vegas algorithms run different constant restart strategies because it is impossible to know the optimal c^*, an SLS process runs them because it does not know which approximation o is reachable within a fixed time t, and every such o might be associated to a different optimal constant restart strategy. This remark further motivates the use of strategy $\mathcal{L}$ for restarting an SLS optimizer.

[3]Note that these probabilities are nondecreasing with o since the best cost value encountered can only improve over time.

4 Multicriteria Strategies

In this section we extend strategy $\mathcal{L}$ to become a multi-criteria strategy, that is, a restart strategy which specifies *what* combination of criteria to optimize and for how long. We assume throughout a d-dimensional cost function $f = (f^1, \ldots, f^d)$ convertible into a one-dimensional function f_λ associated with a weight vector λ ranging over a bounded set Λ of a total volume V.

Definition 8 (Multicriteria Strategy) *A multi-criteria search strategy is an infinite sequence of pairs*

$$S = (t(1), \lambda(1)), (t(2), \lambda(2)), \ldots$$

where for every i, $t(i)$ is a positive integer and $\lambda(i) \in \Lambda$ is a weight vector.

The intended meaning of such a strategy is to run an SLS process to optimize $f_{\lambda(1)}$ for $t(1)$ steps, then $f_{\lambda(2)}$ for $t(2)$ and so on. Following such a strategy and maintaining the set of non-dominated solutions encountered along the way yields an approximation of the Pareto front of f.

Had Λ been a finite set, one could easily adapt the notion of simulation from the previous section and devise a strategy which simulates with a reasonable delay any constant strategy (λ, c) for any $\lambda \in \Lambda$. However since Λ is infinite we need a notion of *approximation*. Looking at two optimization processes, one for f_λ and one for $f_{\lambda'}$ where λ and λ' are close to each other, we observe that the functions may not be very different and the effort spent in optimizing f_λ is almost in the *same direction* as optimizing $f_{\lambda'}$. This motivates the following definition.

Definition 9 (ϵ-Approximation) *A strategy S ϵ-approxi- mates a strategy S' if for every i, $t(i) = t'(i)$ and $|\lambda(i) - \lambda'(i)| < \epsilon$.*

From now on we are interested in finding a strategy which simulates with good delay an ϵ-approximation of any constant strategy (λ, c). To build such a ϵ-*universal* strategy we construct an ε-net D_ε for Λ, that is, a minimal subset of Λ such that for every $\lambda \in \Lambda$ there is some $\mu \in D_\varepsilon$ satisfying $|\lambda - \mu| < \varepsilon$. In other words, D_ε consists of ε-representatives of all possible optimization directions. The cardinality of D_ε depends on the metric used and we take it to be[4] $m_\epsilon = V(1/\varepsilon)^d$. Given D_ε we can create a strategy which is a cross product of $\mathcal{L}$ with D_ε, essentially interleaving m_ϵ instances of $\mathcal{L}$. Clearly, every $\lambda \in \Lambda$ will have at least $1/m_\epsilon$ of the elements in the sequence populated with ε-close values.

Definition 10 (Strategy $\mathcal{L}_{D_\epsilon}$) *Let D be a finite subset of Λ admitting m elements. Strategy $\mathcal{L}_D = ((t(1), \lambda(1)), \ldots$ is defined for every i as*

1. $\lambda(i) = \lambda_{(i \mod m)}$
2. $t(i) = \mathcal{L}(\lceil \frac{i}{m} \rceil)$

Proposition 2 ($\mathcal{L}_{D_\epsilon}$ delay) *Let D_ϵ be an ε-net for Λ. Then $\mathcal{L}_{D_\epsilon}$ simulates an ϵ-approximation of any constant strategy (λ, c) with delay $\delta(t) \leq t m_\epsilon(\lfloor \log t \rfloor + 1)$.*

Proof 2 (Sketch) *[Sketch] For any constant (λ, c) there is an ϵ-close $\mu \in D_\epsilon$ which repeats every m_ϵ^{th} time in $\mathcal{L}_{D_\epsilon}$. Hence the delay of $\mathcal{L}_{D_\epsilon}$ with respect to $\mathcal{L}$ is at most $m_\epsilon t$ and combined with the delay $t(\lfloor \log t \rfloor + 1)$ of $\mathcal{L}$ wrt any constant strategy we obtain the result.*

For a given ε, $\mathcal{L}_{D_\varepsilon}$ is optimal as the following result shows.

Proposition 3 ($\mathcal{L}_{D_\epsilon}$ Optimality) *Any strategy that ϵ-simu
lates every constant strategy has delay $\delta(t) \geq m_\epsilon t/2(\lfloor \log t \rfloor/2 + 1)$ with respect to each of those.*

Proof 3 (Sketch) *Consider such a multicriteria strategy and t steps spent in that strategy. Let $S_{i,j}$ denote the multicriteria constant strategy $(\lambda_i, 2^j), (\lambda_i, 2^j) \ldots$ for all $\lambda_i \in D_\epsilon$ and $j \in \{0..\lfloor \log t \rfloor\}$. The minimum delay when simulating all $S_{i,j}$ for a fixed i is $t/2(\lfloor \log t \rfloor/2 + 1)$ (see proposition 5 in appendix). Because any two $\lambda_i, \lambda_i' \in D_\epsilon$ do not approximate each other, the delays for simulating constant strategies associated with different directions just accumulate. Hence $\delta(t) \geq m_\epsilon t/2(\lfloor \log t \rfloor/2 + 1)$.*

[4]Using other metrics the cardinality may be related to lower powers of $1/\varepsilon$ but the growth is at least linear.

Despite these results, the algorithm has several drawbacks. First computing and storing elements of an ε-net in high dimension is not straightforward. Secondly, multi-dimensional functions of different cost landscapes may require different values of ε in order to explore their Pareto fronts effectively and such an ε cannot be known in advance. In contrast, strategy $\mathcal{L}_D$ needs a different D_ε for each ε with D_ε growing as ε decreases. In fact, the only strategy that can be universal for every ε is a strategy where D is the set of all rational elements of Λ. While such a strategy can be written, its delay is, of course, unbounded.

For this reason, we propose a *stochastic* restart strategy which, for any ϵ, ϵ-simulates all constant multi-criteria strategies with a good expected delay.Our stochastic strategy $\mathcal{L}^r$ is based on the fixed sequence of durations $\mathcal{L}$ and on random sequences of uniformly-drawn elements of Λ.

Definition 11 (Strategy $\mathcal{L}^r$)

- *A stochastic multi-criteria strategy is a probability distribution over multi-criteria strategies;*
- *Stochastic strategy $\mathcal{L}^r$ generates strategies of the form*

$$(t(1), \lambda(1)), (t(2), \lambda(2)), \ldots$$

where $t(1), t(2), \ldots$ is $\mathcal{L}$ and each $\lambda(i)$ is drawn uniformly from Λ.

Note that for any ϵ and λ the probability of an element in the sequence to be ϵ-close to λ is $1/m_\epsilon$. Let us try to give the intuition why for any $\epsilon > 0$ and constant strategy (λ, c), $\mathcal{L}^r$ probabilistically behaves as $\mathcal{L}_{D_\epsilon}$ in the limit. Because each $\lambda(i)$ is drawn uniformly, for any $\epsilon > 0$ the expected number of times $\lambda(i)$ is ϵ-close to λ is the same for any $\lambda \in \Lambda$. So the time is equally shared for ϵ-simulating different directions. Moreover the same time is spent on each constant c as we make use of the time sequence $\mathcal{L}$. Consequently $\mathcal{L}^r$ should ϵ-simulate fairly every strategy (λ, c). We have not yet computed the *expected* delay with which a given multicriteria strategy is simulated by $\mathcal{L}^r$.[5] Nonetheless a weaker reciprocal result directly follows: on a prefix of $\mathcal{L}^r$ of length $tm_\epsilon(\lfloor \log t \rfloor + \log m_\epsilon + 1)$, the expected amount of time ε-spent on any multi-criteria constant strategy (λ, c) is t.

Proposition 4 ($\mathcal{L}^r$ Expected Efficiency) *for all $\epsilon > 0$, after a time $T = tm_\epsilon(\lfloor \log t \rfloor + \log m_\epsilon + 1)$ in strategy $\mathcal{L}^r$, the random variable $w_{\lambda,c}(T)$ of the time spent on ϵ-simulating any constant strategy (λ, c) verifies $\mathbb{E}[w_{\lambda,c}(T)] = t$.*

Proof 4 (Sketch) *[Sketch] In time T, $\mathcal{L}^r$ executes tm_ϵ times any constant c (proposition 1). On the other hand the expected fraction of that time spent for a particular λ is $1/m_\epsilon$.*

5 Experiments

5.1 Quadratic Assignment Problem

The quadratic assignment problem (QAP) introduced in [KB57] is a hard unconstrained combinatorial optimization problem with many real-life applications related to the spatial layout of hospitals, factories or electrical circuits. An instance of the problem consists of n facilities whose mutual interaction is represented by an $n \times n$ matrix F with F_{ij} characterizing the quantity of material flow from facility i to j. In addition there are n locations with mutual distances represented by an $n \times n$ matrix D. A solution is a bijection from facilities to locations whose cost corresponds to the total amount of operational work, which for every pair of facilities is the product of their flow with the distance between their respective locations. Viewing a solution as a permutation π on $\{1, \ldots, n\}$ the cost is formalized as

$$C(\pi) = \sum_{i=1}^{n} \sum_{j=1}^{n} F_{ij} \cdot D_{\pi(i),\pi(j)}.$$

The problem is NP-complete, and even approximating the minimal cost within some constant factor is NP-hard [SG76].

[5] It involves computing the expectation of the delay of $\mathcal{L}$ applied to a random variable with negative binomial distribution.

5.1.1 QAP SLS Design

We implemented an SLS-based solver for the QAP, as well as multi-criteria versions described in the preceding section. The search space for the solver on a problem of size n is the set of permutations of $[1..n]$. We take the neighborhood of a solution π to be the set of solutions π' that can be obtained by swapping two elements. This the kind of move is quite common when solving QAP problems with a local search algorithm. Our implementation also makes use of a standard *incremental* algorithm [Tai91] for maintaining the costs of all neighbors of the current point, which we briefly recall here. Given an initial point, we compute (in cubic time) and store the cost of all its neighbors . After swapping two elements (i, j) of the permutation, we compute the effect of swapping i with j on all costs in the neighborhood. Since we have stored the previous costs, adding the effect of a given swap to the cost of another swap gives a new cost which is valid under the new permutation. This incremental operation can be performed in amortized constant time for each swap, and there are quadratically many possible swaps, resulting in a quadratic algorithm for finding an optimal neighbor.

Algorithm 1 Greedy Randomized Local Search

```
if $rnd() \geq p$ then
    $optimal\_move()$
else
    $rnd\_move()$
end if
```

Concerning the search method, we choose to use a simple greedy selection randomized by noise (algorithm 1). This algorithm is very simple but, as will be shown in the sequel, is quite competitive in practice. The selection mechanism works as follows: with probability $1 - p$ the algorithm makes an optimal step (ties between equivalent neighbors are broken at random), otherwise it goes to a randomly selected neighbor. Probability p can be set to adjust the *noise* during the search. Note that we have concentrated our effort on the comparison of different adaptive strategies for the weight vector rather than on the performance comparison of algorithm 1 with the others classical local search methods (simulated annealing, tabu search).

5.1.2 Experimental Results on QAP library

The QAPLIB[BKR97] is a standard set of benchmarks of one dimensional quadratic assignment problems. Each problem in the set comes with an optimal or best known value. We ran algorithm 1 (implemented in C) using various constant restart strategies and the Luby strategy $\mathcal{L}$ on each of the 134 instances of the library. The machine used for the experiments has an Intel Xeon 3.2GHz processor. Complete results of the simulations reported in table 1. Within a time limit of 500 seconds (for each instance) the algorithm finds the best known value for 93 out of 134 instances. On the remaining problems the average error percentage is of 0.8 %. Also the convergence is fast on small instances ($n \leq 20$) as most of them are brought to optimality in less than 1 second of computation.

Figure 1 plots the number of problems brought to optimal or best-known values as a function of time for each strategy. Among all constant restart values we tried (including infinity) none of them is able to get a better score than strategy $\mathcal{L}$ (at least with a time bound of 500 seconds). This fact corroborates the idea that strategy $\mathcal{L}$ is *universal*, in the sense that the multiplexing of several constant restart values makes it possible to solve (slightly) more instances. This is however done with some *delay* as observed on Figure 1 where strategy $\mathcal{L}$ is outperformed by big constant restarts (1024,10000,∞) at the begining.

5.2 Multi-objective QAP

The multiobjective QAP (mQAP), introduced in [KC03a] allows for multiple flow matrices $F_1, F_2, \ldots F_d$. Each pair (F_i, D) for i in $\{1..d\}$ defines a cost function as presented in 5.1, what renders the problem multiobjective.

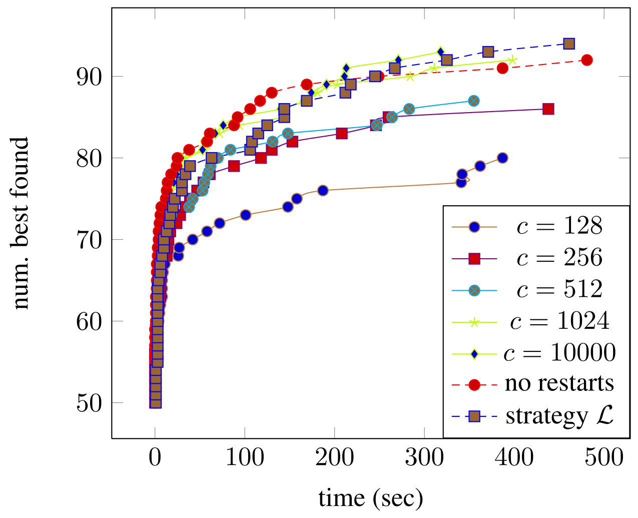

Figure 1: Results of different restart strategies on the 134 QAPLIB problems with optimal or best known results available. Strategy $\mathcal{L}$ solves more instances than the constant restart strategies, supporting the idea that it is efficient in the absence of inside knowledge.

We have developed an extension of algorithm 1 where multiple costs are agglomerated with weighted sums and the set of all non-dominated points encountered during the search is stored in an appropriate structure.

In order to implement strategy $\mathcal{L}$ one also need to generate d-dimensional random weight vectors which are uniformly distributed. Actually, generating random weight vectors is equivalent to sampling uniformly random points on a unit simplex. But, sampling from the d-dimensional unit simplex is also equivalent to sampling from a Dirichlet distribution where every parameter is equal to one. The procedure for generating random weight vectors is therefore the following : 1. Generate d IID random samples $(a_1, \ldots, a_d)$ from a unit-exponential distribution (which is done by sampling a_i from (0,1] uniformly, and returning $-log(a_i)$). 2. Normalize the vector thus obtained by dividing each coordinate by the sum $S = \sum_{i=0}^{d} a_i$.

Multi-criteria strategies $\mathcal{L}^r$ and $\mathcal{L}_{D_\epsilon}$ have been tested and compared on the benchmarks of the mQAP library, which includes 2-dimensional problems of size n = 10 and n = 20, and 3-dimensional problems of size n = 30 [KC03b]. The library contains both instances generated uniformly at random, and instances generated according to flow and distance parameters which reflects the probability distributions found in real situations. Also, pre-computed Pareto sets are provided for all 10-facility instances. We should note that our algorithm found over 99% of all Pareto points from all eight 10-facility problems, within a *total* computation time of under 1 second.

For the remaining problems where the actual Pareto set is unknown, we resort to comparing performance against the non-dominated union of solutions found with any configuration. As a quality indicator we use the ϵ-approximation concept [ZKT08] which indicates the largest *increase* in cost of a solution in the reference set, when compared to its best approximation in the set to evaluate. The errors are normalized with respect to the difference between the maximal and minimal costs in each dimension over all samples of restart strategies leading to a number $\alpha \in [0, 1]$ indicating the error with respect to the set of all found solutions.[6]

Figures 2, 3 and 4 depict the results of the multi-criteria experiments. We compared $\mathcal{L}^r$ against constant restarts combined with randomly chosen directions and strategy $\mathcal{L}_{D_\varepsilon}$ for different values of ε. Despite its theoretical properties $\mathcal{L}_{D_\varepsilon}$ does not perform so good for the ε values that we chose. This corroborates the fact that it is hard to guess a good ϵ value beforehand. Constantly restarting works good but the appropriate constant has to be carefully chosen. At the end strategy $\mathcal{L}^r$ gives decidedly *better performance amongst all*

[6]In the future we also intend to use techniques which can formally validate the quality of a Pareto set approximation [LGCM10] in order to assess the efficiency of this algorithm.

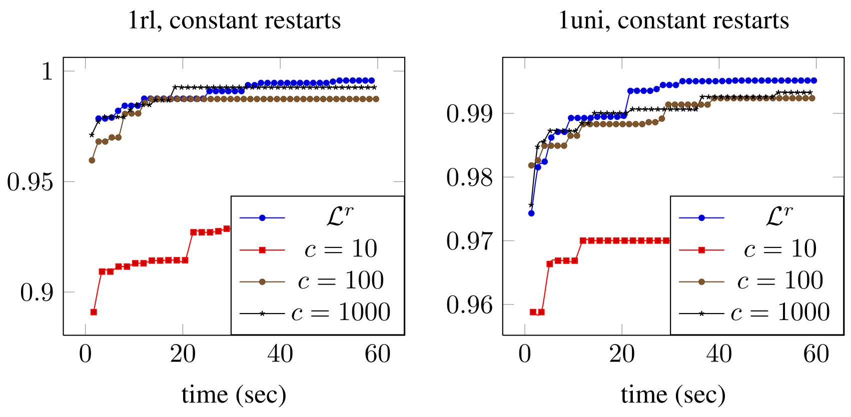

Figure 2: Performance of various constant multi-criteria strategies against $\mathcal{L}^r$ on the 1rl and 1uni 20-facility problems. The metric used is the normalized ϵ-indicator, the reference set being the union of all solutions found. In these experiments constant restarts values $c = 10, 100, 1000$ are combined with randomly generated directions.

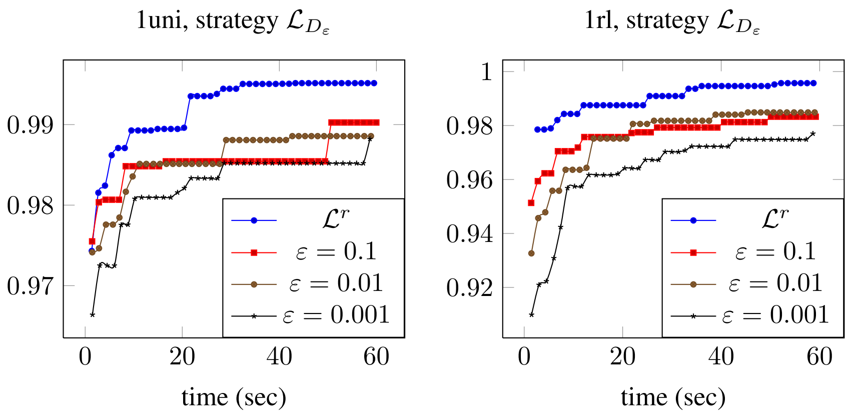

Figure 3: Performance of $\mathcal{L}_{D_\varepsilon}$ multi-criteria strategies against $\mathcal{L}^r$ on the 1rl and 1uni 20-facility problems for $\varepsilon = 0.1, 0.01, 0.001$, mesured by the ϵ-indicator metric.

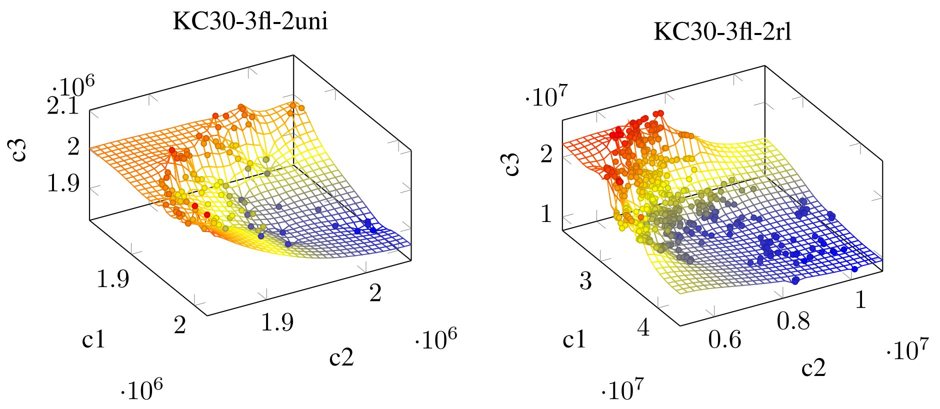

Figure 4: Examples of approximations of the Pareto front for two 30-facility, 3-flow QAP problems. KC30-3fl-2uni was generated uniformly at random while KC30-3fl-2rl was generated at random with distribution parameters, such as variance, set to reflect real instances. The approximations above are a result of 1 minute of computation using $\mathcal{L}^r$ as multi-criteria strategy.

the strategies we tried. It is worth noting that we also tried using the weighted infinity norm $\max_i \lambda_i f^i$ as a measure of cost but, unless the theoretical interest (Pareto points on concave parts of the front are optimal for some valuation of the weights), the method did perform worse than the weighted sum approach on our experiments.

6 Conclusion

We have demonstrated how efficient universal strategies can accelerate SLS-based optimization, at least in the absence of knowledge of good restart constants. In the multi-criteria case, our approximating universal strategy is efficient, both theoretically and experimentally and gives a thorough and balanced coverage of the Pareto front. After having demonstrated the strength of our algorithm on the QAP benchmarks, we are now working on two extensions of this work. First, we are planning to use strategy $\mathcal{L}^r$ with other efficient local search techniques, starting with tabu search [Tai91]. Secondly we intend to move on to problems associated with mapping and scheduling of programs on multi-processor architectures and see how useful this algorithmics can be for those problems.

References

[AC91] M. Dorigo et V. Maniezzo A. Colorni. Distributed optimization by ant colonies. In *First European Conference on Artificial Life*, pages 134–142. Elsevier, 1991. 2.1, 2.3

[BKR97] R.E. Burkard, S.E. Karish, and F. Rendl. Qaplib – a quadratic assignment problem library. *Journal of Global Optimization*, 10:391–403, 1997. 5.1.2

[BSMD08] S. Bandyopadhyay, S. Saha, U. Maulik, and K. Deb. A simulated annealing-based multiobjective optimization algorithm: AMOSA. *Evolutionary Computation, IEEE Transactions on*, 12(3):269–283, 2008. 2.3

[CJ98] P. Czyzak and A. Jaszkiewicz. Pareto simulated annealing-a metaheuristic technique for multiple-objective combinatorial optimization. *Journal of Multi-Criteria Decision Analysis*, 7(1):34–47, 1998. 2.3

[Deb01] K. Deb. *Multi-objective optimization using evolutionary algorithms*. Wiley, 2001. 2.2, 2.3

[Ehr05] M. Ehrgott. *Multicriteria optimization*. Springer Verlag, 2005. 2.2, 2.2

[FR95] T.A. Feo and M.G.C. Resende. Greedy randomized adaptive search procedures. *Journal of Global Optimization*, 6(2):109–133, 1995. 2.3

[Geo67] A.M. Geoffrion. Proper Efficiency and the Theory of Vector Maximization. 1967. 2.2

[GM06] F. Glover and R. Marti. Tabu search. *Metaheuristic Procedures for Training Neutral Networks*, pages 53–69, 2006. 2.3

[GMF97] X. Gandibleux, N. Mezdaoui, and A. Fréville. A Tabu Search Procedure to Solve Multiobjective Combinatorial Optimization Problem. *Lecture notes in economics and mathematical systems*, pages 291–300, 1997. 2.3

[Han97] M.P. Hansen. Tabu search for multiobjective optimization: MOTS. In *Proc. 13th Int. Conf/ on Multiple Criteria Decision Making (MCDM)*, pages 574–586, 1997. 2.3

[Hel09] K. Helsgaun. General k-opt submoves for the lin-kernighan tsp heuristic. *Mathematical Programming Computation*, 2009. 2.1

[HS04] H. Hoos and T. Stützle. *Stochastic Local Search Foundations and Applications*. Morgan Kaufmann / Elsevier, 2004. 2.1, 2.3

[KB57] T.C. Koopmans and M. Beckman. Assignment problems and the location of economic activities. *Econometric*, 25:53–76, 1957. 5.1

[KC03a] J. Knowles and D. Corne. Instance generators and test suites for the multiobjective quadratic assignment problem. In C. Fonseca, P. Fleming, E. Zitzler, K. Deb, and L. Thiele, editors, *Evolutionary Multi-Criterion Optimization*, number 2632 in LNCS, pages 295–310. Springer, 2003. 5.2

[KC03b] J. Knowles and D. Corne. Instance generators and test suites for the multiobjective quadratic assignment problem. pages 72–72, 2003. 5.2

[KC05] J. Knowles and D. Corne. Memetic algorithms for multiobjective optimization: issues, methods and prospects. *Recent advances in memetic algorithms*, pages 313–352, 2005. 2.3

[KGV83] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, New Series 220 (4598):671–680, 1983. 2.1, 2.3

[LGCM10] Julien Legriel, Colas Le Guernic, Scott Cotton, and Oded Maler. Approximating the pareto front of multi-criteria optimization problems. In *TACAS*, pages 69–83, 2010. 6

[LMS03] H. Lourenco, O. Martin, and T. Stützle. Iterated local search. *Handbook of metaheuristics*, pages 320–353, 2003. 2.3

[LSZ93] M. Luby, A. Sinclair, and D. Zuckerman. Optimal speedup of las vegas algorithms. In *Israel Symposium on Theory of Computing Systems*, pages 128–133, 1993. (document), 1, 2.3, 3, 3.1, 3.1, 6

[Mat65] J. Matyas. Random optimization. *Automation and Remote Control*, 2(26):246–253, 1965. 2.1

[Mit98] M. Mitchell. *An introduction to genetic algorithms*. The MIT press, 1998. 2.3

[Par12] V. Pareto. Manuel d'économie politique. *Bull. Amer. Math. Soc.*, 18:462–474, 1912. 1

[PD07] K. Pipatsrisawat and A. Darwiche. A Lightweight Component Caching Scheme for Satisfiability Solvers. In *SAT-07*, 2007. 1

[PS06] L. Paquete and T. Stützle. Stochastic local search algorithms for multiobjective combinatorial optimization: A review. Technical Report TR/IRIDIA/2006-001, IRIDIA, 2006. 2.3

[SG76] S. Sahni and T. Gonzalez. P-complete approximation problems. *Journal of the ACM*, 23:555–565, 1976. 5.1

[SKC93] B. Selman, H. Kautz, and B. Cohen. Local Search Strategies for Satisfiability Testing. In D. S. Johnson and M. A. Trick, editors, *Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge*, October 1993. 2.1, 3

[Tai91] E. Taillard. Robust taboo search for the quadratic assignment problem. *Parallel computing*, 17(4-5):443–455, 1991. 5.1.1, 6

[TH05] D. Tompkins and H. Hoos. UBCSAT: An implementation and experimentation environ- ment for SLS algorithms for SAT and MAX-SAT. In *SAT-04*, volume 3542 of *LNCS*, pages 306–320, 2005. 2.1

[UTFT99] EL Ulungu, J. Teghem, PH Fortemps, and D. Tuyttens. MOSA method: a tool for solving multiobjective combinatorial optimization problems. *Journal of Multi-Criteria Decision Analysis*, 8(4):221–236, 1999. 2.3

[ZKT08] E. Zitzler, J. Knowles, and L. Thiele. Quality assessment of Pareto set approximations. *Multiobjective Optimization*, pages 373–404, 2008. 5.2

Appendix

Optimality of $\mathcal{L}$

The optimality of $\mathcal{L}$ is proved in [LSZ93]. We reformulate it using the notion of delay introduced in this paper and give a sketch of proof.

Proposition 5 ($\mathcal{L}$ Optimality) *Any time strategy which simulates every constant time strategy does it with delay $\delta(t) \geq t/2(\lfloor \log t \rfloor/2 + 1)$.*

Proof 5 (Sketch) *let S be such a strategy and t a time. Consider the constant restart strategies $\{S_i = 2^i, 2^i \dots\}_0^{\lfloor logt \rfloor}$. Each t-prefix of these strategies must be simulated by a $\delta(t)$-prefix of S. In particular the t-prefix of $S_{\lfloor logt \rfloor}$ which is $2^{\lfloor logt \rfloor}$ has to be simulated. So the $\delta(t)$-prefix of S needs an element bigger than or equal to $2^{\lfloor logt \rfloor}$. Also the t-prefix of $S_{\lfloor logt \rfloor - 1}$ which is $2^{\lfloor logt \rfloor - 1}, 2^{\lfloor logt \rfloor - 1}$ must be simulated. One of the two values can be mapped to the value greater than $2^{\lfloor logt \rfloor}$, but there should be another value greater than $2^{\lfloor logt \rfloor - 1}$. From the previous observations we have that $\delta(t) \geq 2^{\lfloor logt \rfloor} + 2^{\lfloor logt \rfloor - 1}$. If we denote N_i the minimum number of 2^i values that must be present in the $\delta(t)$-prefix of S we can formulate this as 1. $N_{\lfloor logt \rfloor} = 1$ and 2. $\forall i \leq \lfloor logt \rfloor - 1$, $N_i = 2^{\lfloor logt \rfloor - i} - \sum_{j=i+1}^{\lfloor logt \rfloor - 1} N_j - 1$ (i.e for each i values bigger than 2^i can be used to simulate a 2^i). By recursion we can show that $\forall i \leq \lfloor logt \rfloor - 1,\ N_i = 2^{\lfloor logt \rfloor - i - 1}$. We therefore get $\delta(t) \geq 2^{\lfloor logt \rfloor} + \sum_{i=0}^{\lfloor logt \rfloor - 1} 2^{\lfloor logt \rfloor - 1 - i}\ 2^i$. After simplifying the sum we obtain $\delta(t) \geq t/2(\lfloor \log t \rfloor/2 + 1)$.*

Single-criterium QAP Results

Table 1 shows the results and execution times obtained by our SLS algorithm on QAPLIB problems, compared to the best known results.

name	size	best value	mqap best	steps	time	name	size	best value	mqap best	steps	time
bur26a	26	5426670	5426670	87333	10.16	bur26b	26	3817852	3817852	25874	3.00
bur26c	26	5426795	5426795	52541	6.10	bur26d	26	3821225	3821225	4146	0.49
bur26e	26	5386879	5386879	26507	3.12	bur26f	26	3782044	3782044	9485	1.14
bur26g	26	10117172	10117172	26016	3.03	bur26h	26	7098658	7098658	9363	1.09
chr12a	12	9552	9552	10134	0.23	chr12b	12	9742	9742	1782	0.04
chr12c	12	11156	11156	19000	0.43	chr15a	15	9896	9896	5924	0.22
chr15b	15	7990	7990	91494	3.36	chr15c	15	9504	9504	88003	3.23
chr18a	18	11098	11098	196208	10.53	chr18b	18	1534	1534	5654	0.31
chr20a	20	2192	2192	3650116	245.31	chr20b	20	2298	2298	2154279	144.99
chr20c	20	14142	14142	205534	13.76	chr22a	22	6156	6156	2624486	212.39
chr22b	22	6194	6272	6182333	500.00	chr25a	25	3796	3822	4671871	500.00
esc128	128	64	64	937	3.44	esc16a	16	68	68	122	0.00
esc16b	16	292	292	40	0.00	esc16c	16	160	160	154	0.00
esc16d	16	16	16	43	0.00	esc16e	16	28	28	112	0.00
esc16f	16	0	0	1	0.00	esc16g	16	26	26	47	0.00
esc16h	16	996	996	42	0.00	esc16i	16	14	14	16	0.00
esc16j	16	8	8	16	0.00	esc32a	32	130	130	169182	30.31
esc32b	32	168	168	5063	0.95	esc32c	32	642	642	242	0.04
esc32d	32	200	200	1144	0.20	esc32e	32	2	2	11	0.00
esc32f	32	2	2	11	0.00	esc32g	32	6	6	14	0.00
esc32h	32	438	438	894	0.17	esc64a	64	116	116	110	0.10
had12	12	1652	1652	383	0.00	had14	14	2724	2724	1065	0.04
had16	16	3720	3720	702	0.03	had18	18	5358	5358	2975	0.16
had20	20	6922	6922	1477	0.10	kra30a	30	88900	88900	685946	106.45
kra30b	30	91420	91420	2981280	461.82	kra32	32	88900	26604	1	0.00
lipa20a	20	3683	3683	7321	0.50	lipa20b	20	27076	27076	1162	0.08
lipa30a	30	13178	13178	144126	22.49	lipa30b	30	151426	151426	11909	1.91
lipa40a	40	31538	31538	376838	108.15	lipa40b	40	476581	476581	15751	4.56
lipa50a	50	62093	62684	1087569	500.00	lipa50b	50	1210244	1210244	44406	20.99
lipa60a	60	107218	108152	738889	500.00	lipa60b	60	2520135	2520135	58474	39.91
lipa70a	70	169755	171057	524491	500.00	lipa70b	70	4603200	4603200	232953	218.46
lipa80a	80	253195	254942	398776	500.00	lipa80b	80	7763962	7763962	264266	325.77
lipa90a	90	360630	362964	314604	500.00	lipa90b	90	12490441	12490441	89043	144.09
nug12	12	578	578	126	0.00	nug14	14	1014	1014	14557	0.46
nug15	15	1150	1150	2120	0.08	nug16a	16	1610	1610	30191	1.29
nug16b	16	1240	1240	1984	0.08	nug17	17	1732	1732	16209	0.76
nug18	18	1930	1930	118373	6.39	nug20	20	2570	2570	8685	0.58
nug21	21	2438	2438	47793	3.59	nug22	22	3596	3596	1696	0.14
nug24	24	3488	3488	16016	1.58	nug25	25	3744	3744	81772	8.84
nug27	27	5234	5234	275874	34.38	nug28	28	5166	5166	26105	3.61
nug30	30	6124	6124	1093652	169.83	rou12	12	235528	235528	3259	0.07
rou15	15	354210	354210	4021	0.15	rou20	20	725522	725522	264911	17.90
scr12	12	31410	31410	2145	0.04	scr15	15	51140	51140	3511	0.13
scr20	20	110030	110030	14984	1.02	sko100a	100	152002	152696	250279	500.00
sko100b	100	153890	154890	253747	500.00	sko100c	100	147862	148614	253422	500.00
sko100d	100	149576	150634	253601	500.00	sko100e	100	149150	150004	253401	500.00
sko100f	100	149036	150032	254968	500.00	sko42	42	15812	15852	1552963	500.00
sko49	49	23386	23474	1146362	500.00	sko56	56	34458	34602	856498	500.00
sko64	64	48498	48648	631733	500.00	sko72	72	66256	66504	500701	500.00
sko81	81	90998	91286	391585	500.00	sko90	90	115534	116192	309691	500.00
ste36a	36	9526	9586	2163008	500.00	ste36b	36	15852	15852	494591	115.02
ste36c	36	8239110	8249952	2195600	500.00	tai100a	100	21125314	21591086	255199	500.00
tai100b	100	1185996137	1193503855	254527	500.00	tai12a	12	224416	224416	45	0.00
tai12b	12	39464925	39464925	2496	0.06	tai150b	150	498896643	507195710	101380	500.00
tai15a	15	388214	388214	1136	0.04	tai15b	15	51765268	51765268	438	0.02
tai17a	17	491812	491812	76666	3.68	tai20a	20	703482	703482	960799	63.94
tai20b	20	122455319	122455319	8079	0.56	tai256c	256	44759294	44866220	28941	500.00
tai25a	25	1167256	1171944	4729773	500.00	tai25b	25	344355646	344355646	285069	30.29
tai30a	30	1818146	1828398	3236381	500.00	tai30b	30	637117113	637117113	1722202	267.30
tai35a	35	2422002	2462770	2334384	500.00	tai35b	35	283315445	283315445	571374	125.05
tai40a	40	3139370	3190650	1763043	500.00	tai40b	40	637250948	637250948	56899	16.74
tai50a	50	4941410	5067502	1083719	500.00	tai50b	50	458821517	458966955	1048395	500.00
tai60a	60	7208572	7392986	739306	500.00	tai60b	60	608215054	608758153	742214	500.00
tai64c	64	1855928	1855928	3636	3.02	tai80a	80	13557864	13846852	407800	500.00
tai80b	80	818415043	825511580	404012	500.00	tho150	150	8133398	8215216	101551	500.00
tho30	30	149936	149936	2396680	371.55	tho40	40	240516	241598	1729732	500.00
wil100	100	273038	273854	253337	500.00	wil50	50	48816	48862	1071145	500.00

Table 1: Results of running algorithm 1 on the QAP library instances. Each row shows the best value obtained within a time limit of 500 seconds, compared to the best known or optimal value.