# Optimization of the Nested Monte-Carlo Algorithm on the Traveling Salesman Problem with Time Windows

Arpad Rimmel[1], Fabien Teytaud[2], and Tristan Cazenave[1]

[1] LAMSADE, Université Paris Dauphine, France,
[2] TAO (Inria), LRI, Univ. Paris-Sud, France

**Abstract.** The traveling salesman problem with time windows is known to be a really difficult benchmark for optimization algorithms. In this paper, we are interested in the minimization of the travel cost. To solve this problem, we propose to use the nested Monte-Carlo algorithm combined with a Self-Adaptation Evolution Strategy. We compare the efficiency of several fitness functions. We show that with our technique we can reach the state of the art solutions for a lot of problems in a short period of time.

**Keywords:** Nested Monte-Carlo, Self Adaptation, Traveling Salesman, Time Window.

## 1 Introduction

The traveling salesman problem is a difficult optimization problem and is used as a benchmark for several optimization algorithms. In this paper we tackle the problem of optimizing the Traveling Salesman Problem with Time Windows (TSPTW). For solving this problem, we combine a nested Monte-Carlo algorithm [4] and an evolutionary algorithm. With this system, as we will see, the important point is that we will have to optimize a function which is noisy and where the evaluation is not the score on average but the best score among a certain number of runs. When the noise is uniform on the whole function, optimizing for the mean or for the min is equivalent, so we will focus on problems where the noise is non uniform. We will show on an artificial problem that having a fitness function during the optimization that is different from the one we want to optimize can improve the convergence rate. We will then use this principle to optimize the parameters of a nested Monte-Carlo algorithm for the TSPTW.

We have chosen the use of Evolution-Strategies (ES [12]) for the optimization part. This kind of algorithms are known to be simple and robust. See [12, 2] for more details on ES in general.

The paper is organized as follows : Section 2 presents the optimization algorithm used, Section 3 is the presentation of the artificial problem and the results we obtain, Section 4 is the application to the TSPTW, and finally we discuss all the results in the Section 5.

## 2 Self-Adaptation Evolution Strategy

In all the paper we use $(\mu/\mu, \lambda)$-ES. $\lambda$ is the population size, $\mu$ the number of parents (the selected population size), and the parents are selected only among individuals belonging to the new generated population (and not in the mixing between the new generated population and the previous parents). Mutation will be done according to a Gaussian distribution.

We have chosen the Self-Adaptation Evolution Strategy (SA-ES) for the optimization of our problem. This algorithm has been introduced in [12] and [14]. An extended version with full covariance matrix has been proposed in [3]. An improvement of this algorithm, based on the selection ratio, efficient in the case of large population can also be used [16]. In our experiments, we use the standard SA-ES with small population sizes, then we do not use this last improvement. The motivation behind the choice of this algorithm is that it is known to be really robust, because it doesn't need any a priori knowledge on the problem. The SA-ES algorithm is presented in Algorithm 1.

**Algorithm 1** Mutative self-adaptation algorithm.

```
Initialize $\sigma^{avg} \in \mathbb{R}$, $y \in \mathbb{R}^N$.
while Halting criterion not fulfilled do
  for $i = 1..\lambda$ do
    $\sigma_i = \sigma^{avg} e^{\tau N_i(0,1)}$
    $z_i = \sigma_i N_i(0, Id)$
    $y_i = y + z_i$
    $f_i = f(y_i)$
  end for
  Sort the individuals by increasing fitness; $f_{(1)} < f_{(2)} < \cdots < f_{(\lambda)}$.
  $z^{avg} = \frac{1}{\mu} \sum_{i=1}^{\mu} z_{(i)}$
  $\sigma^{avg} = \frac{1}{\mu} \sum_{i=1}^{\mu} \sigma_{(i)}$
  $y = y + z^{avg}$
end while
```

## 3 Noisy Sphere

We will first optimize on an artificial problem: the noisy sphere.

### 3.1 Presentation

The noisy sphere is a classical artificial problem for optimization experiments. However, here, the noise function will be original. The noise will be Gaussian and non uniform. We will use 5 dimensions.

The exact equation of the function that we will use is the following:

$$f(y) = \sum_{i=1}^{N} (y_i^2 + N(0, (2y_i)^2)).$$

It is represented on the top-left of figure 1.

The evaluation function $eval(f(y))$ will be the min over 1000 runs of $f$ with parameters $y$. We will optimize the expectation of this function:

$$eval(f(y)) = \min(f^i(y), i = 1..1000),$$

$f^i$ being the $i$-th run of $f$.

During the optimization, we will use a fitness function to evaluate an individual. Usually, people use the same function for the fitness function and for the evaluation function. However, we see that the function that we want to optimize is very unstable. For this reason, we propose to use a fitness function that can be different from the evaluation function.

The 3 fitness functions that we will use are:

- $best_n(f(y)) = \min(f^i(y), i = 1..n)$
- $mean_n(f(y)) = \frac{\sum_{i=1}^{n} f^i(y)}{n}$
- $kbest_{k,n}(f(y)) = \frac{\sum_{i=1}^{k} f^i(y)}{n}$ with $f^1(y) < f^2(y) < ... < f^k(y) < ... < f^n(y)$

As the fitness function used during the optimization is not the same as the evaluation function, we compute for each generation the score of the best individual according to the evaluation function. This function is noisy, so the **true score** of an individual will be the average over $NbEval$ runs of the evaluation function. This is very costly in number of evaluations but this true score is only used to show that the algorithm converges and will only be used for the noisy sphere.

## 3.2 Experiments

We use the SA algorithm to do the optimizations. In the experiments, we used $k = 5$ for *kbest* and $NbEval = 100$.

We compare $best_n$, $mean_n$ and $kbest_{5,n}$ for different values of $n$.

We compute the true score of the best individual in function of the number of the generation. Every curve is the average over 30 runs.

The results are given on the figure 1.

We see that in every cases, the convergence is slower with *best* than with *mean* and *kbest*. However, the final value is always better for *best*. This is because *best* is the fitness function the most similar to the evaluation function.

For high values of $n$, the convergence is equivalent for *kbest* and *mean*. Furthermore, the final value is better for *kbest* than for *mean*. This implies that for high value of $n$, it is always better to use *kbest* instead of *mean*.

For small values of $n$, *kbest* converges slowly than *mean* but achieves a better final value.

As a conclusion, the choice of the fitness function will depend on the need of the user. If the speed of the convergence is important, one can use *mean* or *kbest* depending on $n$. If the final value is important, *best* is the function to use.

We will now see if the conclusions we obtained on an artificial problem are still valid when we optimize on difficult benchmarks.

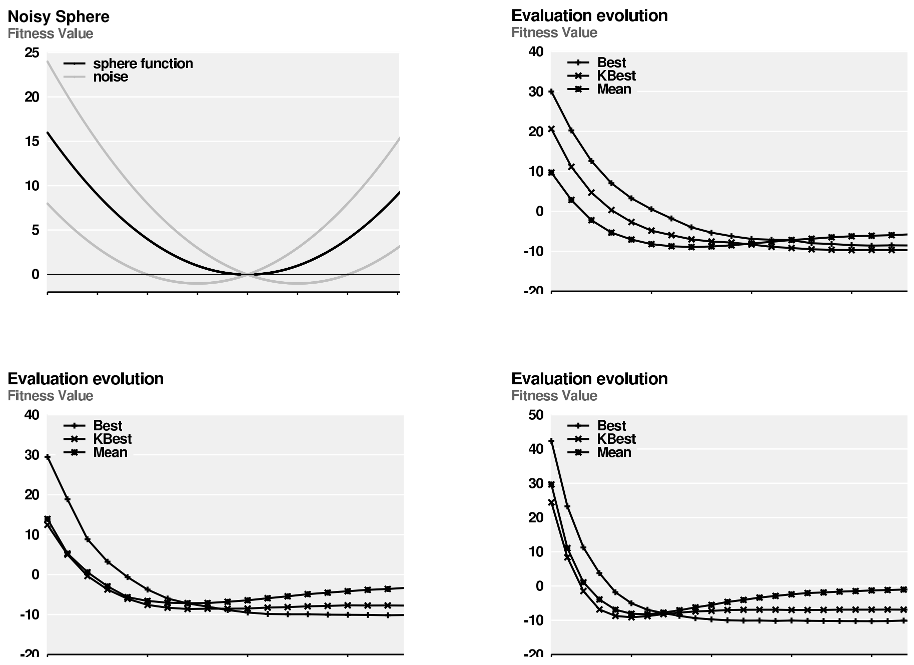

**Fig. 1. Top-left.** Representation of the evaluation function. **Top-right, bottom-left, bottom-right.** Evolution of the true score as a function of the iterations with $n = 10$, $n = 100$ and $n = 300$ respectively.

## 4 Application

We will now focus on the TSPTW problem. First, we will describe the problem. Then, we will present the nested Monte-Carlo algorithm. Finally, we will show the results we obtain when we optimize the parameters of the algorithm on TSPTW.

### 4.1 Traveling Salesman Problem with Time Windows

The traveling salesman problem is an important logistic problem. It is used to represent the problem of finding an efficient route to visit a certain number of customers, starting and finishing at a depot. The version with time windows adds the difficulty that each customer has to be visited within a given period of time. The goal is to minimize the length of the travel. TSPTW is an NP-hard problem and even finding a feasible solution is NP-complete [13]. Early works [5, 1] were based on branch-and-bound. Later, Dumas et al. used a method based on Dynamic programming [6]. More recently, methods based on constraints programming have been proposed [10, 7].

Algorithms based on heuristics have also been considered [15, 8].

Finally, [9] provides a comprehensive survey of the most efficient methods to solve the TSPTW and proposes a new algorithm based on ant colonies that achieves very good results. They provide a clear environment to compare algorithms on a set of problems that we used in this article.

**Technical description of the Traveling Salesman Problem with Time Windows.** The TSP can be described as follow. Let $G$ be an undirected complete graph. $G = (N, A)$ where $N = 0, 1, ..., n$ is a set of nodes and $A = N * N$ is the set of edges between the nodes. The node 0 represents the depot. The $n$ other nodes represent the customers. A cost function $c : A \to \mathbb{R}$ is given. It represents the distance between 2 customers. A solution to this problem is a sequence of nodes $P = (p_0, p_1, ..., p_n, p_{n+1})$ where $p_0 = p_{n+1} = 0$ and $(p_1, ..., p_n)$ is a permutation of $N \setminus \{0\}$.

The goal is to minimize the function

$$cost(P) = \sum_{k=0}^{n} c(a_{p_k,p_{k+1}}).$$

In the version with time windows, each customer $i$ is associated with an interval $[e_i, l_i]$. The customer must not be served before $e_i$ or after $l_i$. It is allowed to arrive at a node $i$ before $e_i$ but the departure time becomes $e_i$.

Let $d_{p_k}$ be the departure time from node $p_k$, $d_{p_k} = \max(r_{p_k}, e_{p_k})$ where $r_{p_k}$ is the arrival time at node $p_k$.

The function to minimize is the same but a set of constraints must now be respected. Let $\Omega(P)$ be the number of windows constraints violated by tour P. The optimization of $f$ must be done while respecting the following equation

$$\Omega(P) = \sum_{k=0}^{n+1} \omega(p_k) = 0,$$

where

$$\omega(p_k) = \begin{cases} 1 \text{ if } r_{p_k} > l_{p_k} \\ 0 \text{ otherwise} \end{cases},$$

and

$$r_{p_{k+1}} = \max(r_{p_k}, e_{p_k}) + c(a_{p_k,p_{k+1}}).$$

With the addition of the constraints, the problem becomes much more complicated and classical algorithms used for TSP are not efficient anymore. That is why we will use Nested Monte-Carlo which is described in the next part of the article.

## 4.2 Adaptation of the Nested Monte-Carlo Algorithm for the Traveling Salesman Problem with Time Windows

The Nested Monte-Carlo (NMC) algorithm [4] is a tree search algorithm. The tree is supposed to be large and the leaves of the tree (final positions of the problem) can be evaluated. It does not require any knowledge on the problem and is quite simple to implement. It is particularly efficient on problems where later decisions are as important as early ones. NMC has allowed to establish world records in single-player games such as Morpion Solitaire or SameGame. We first describe the NMC algorithm and then explain how we introduced heuristics in order to obtain better results on the TSPTW problem.

**The Nested Monte-Carlo Algorithm.** The NMC algorithm uses several levels. Each level uses the lower level to determine which action will be selected at each step. The level 0 is a Monte-Carlo simulation, i.e. a random selection of actions until a final position is reached. More precisely, in each position, a NMC search of level $n$ will perform a level $n-1$ NMC for each action and then select the one with the best score. For example, a NMC search of level 1 will do a Monte-Carlo simulation for each action (those reaching a final position which can be evaluated) and select the action associated with the highest evaluation. Once an action has been selected, the problem is in a new position and the selection method is repeated again until a final position is reached. The performance of the algorithm is greatly improved by memorizing the best sequence for each level.

**Algorithm 2** Nested Monte-Carlo

```
nested($position$, $level$)
best playout ← {}
while not end of the game do
  if $level = 1$ then
    $move$ ← $\arg\max_m$(MonteCarlo(play($position$, $m$)))
  else
    $move$ ← $\arg\max_m$(nested(play($position$, $m$), $level - 1$))
  end if
  if score of playout after $move$ > score of the best playout then
    best playout ← playout after $move$
  end if
  $position$ ← play($position$, move of the best playout)
end while
return score($position$)
```

`play`(*position*, $m$) is a function that returns the new position obtained after having selected the action $m$ in *position*

`MonteCarlo`(*position*) is a function that returns the evaluation of the final position reached after having selected random actions from *position*.

NMC provides a good compromise between exploration and exploitation. It is particularly efficient for one-player games and gives good results even without domain knowledge. However, the results can be improved by the addition of heuristics.

**Adaptation for the Traveling Salesman Problem with Time Windows.** It is possible to improve the performance of NMC by modifying the Monte-Carlo simulations. An efficient way is to select actions based on heuristics instead of a uniform distribution. However, some randomness must be kept in order to preserve the diversity of the simulations.

To do that, we use a Boltzmann softmax policy. This policy is defined by the probability $\pi_\theta(p,a)$ of choosing the action $a$ in a position $p$:

$$\pi_\theta(p,a) = \frac{e^{\phi(p,a)^T\theta}}{\sum_b e^{\phi(p,b)^T\theta}},$$

where $\phi(p,a)$ is a vector of features and $\theta$ is a vector of feature weights.

The features we use are the heuristics described in [15]:

- the distance to the last node: $h1(p,a) = c(d,a)$
- the amount of time that will be necessary to wait if $a$ is selected because of the beginning of its time window: $h2(p,a) = \max(0, e_a - (T_p + c(d,a)))$
- the amount of time left until the end of the time window of $a$ if $a$ is selected: $h3(p,a) = \max(0, l_a - (T_p + c(d,a)))$

where $d$ is the last node selected in position $p$, $T_p$ is the time used to arrive in situation $p$, $e_a$ is the beginning of the time window for action $a$, $l_a$ is the end of the time window for the action $a$ and $c(d,a)$ is the travel cost between $d$ and $a$.

The values of the heuristic are normalized before being used.

The values that we will optimize are the values from the vector $\theta$ (the feature weights).

### 4.3 Experiments

We use the set of problems given in [11].

As we have 3 different heuristics, the dimension of the optimization problem is 3.

We define $NMC(y)$, the function that associates a set of parameters $y$ to the permutation obtained by a run of the NMC algorithm, with parameters $y$ on a particular problem.

The score $Tcost(p)$ of a permutation $p$ is the travel cost. However, as the NMC algorithm can generate permutations with some windows constraints not respected, we added a constant to the score for each one.

$$Tcost(p) = cost(p) + 10^6 * \Omega(p),$$

$cost(p)$ is the cost of the travel $p$ and $\Omega(p)$ the number of non-respected constraints.

$10^6$ is a constant high enough for the algorithm to first optimize $\Omega(p)$ and then $cost(p)$.

The exact equation of the function $f$ that we will use is the following:

$$f(y) = Tcost(NMC(y)).$$

As the end of the evaluation, we want to obtain a NMC algorithm that we will launch for a longer period of time in order to obtain one good score on a problem. So the evaluation function should be the min on this period of time.

As this period of time is not known and a large period of time would be too time-consuming, we arbitrarily choose a time of 1s to estimate the true score of an individual.

The evaluation function $eval(f(y))$ will be the min over $r$ runs of $f$ with parameters $y$. $r$ being the amount of runs that can be done in 1s. It means that we want to optimize the expectation of this function:

$$eval(f(y)) = \min_{1s}(f(y)).$$

As for the sphere problem, we will use 3 different fitness functions instead of the evaluation function: $mean_n$, $kbest_n$ and $best_n$. In the experiments, we use $n = 100$.

We use a nested algorithm of level 2.

**Optimization on one Problem.** The first experiments are done on the problem rc206.3 which contains 25 nodes.

In this experiment we compare $best_{100}$, $kbest_{100}$ and $mean_{100}$. As in all the paper, the population size $\lambda$ is equal to 12 and the selected population size $\mu$ is 3, and $\sigma = 1$. The initial parameters are $[1, 1, 1]$ and the stopping criterion of the evolution-strategy is 15 iterations. Results are the average of three independent runs.

| Iterations | BEST | KBEST | MEAN |
|---|---|---|---|
| 1 | 2.7574e+06 | 2.4007e+06 | 2.3674e+06 |
| 2 | 5.7322e+04 | 3.8398e+05 | 1.9397e+05 |
| 3 | 7.2796e004 | 1.6397e+05 | 618.22 |
| 4 | 5.7274e+04 | 612.60 | 606.68 |
| 5 | 2.4393e+05 | 601.15 | 604.10 |
| 6 | 598.76 | 596.02 | 602.96 |
| 7 | 599.65 | 596.19 | 603.69 |
| 8 | 598.26 | 594.81 | 600.79 |
| 9 | 596.98 | 591.64 | 602.54 |
| 10 | 595.13 | 590.30 | 600.14 |
| 11 | 590.62 | 591.38 | 600.68 |
| 12 | 593.43 | 589.87 | 599.63 |
| 13 | 594.88 | 590.47 | 599.24 |
| 14 | 590.60 | 589.54 | 597.58 |
| 15 | 589.07 | 590.07 | 599.73 |

**Table 1.** Evolution of the true score on the problem rc206.3.

There is a lot of difference between the initial parameters and optimized parameters in term of performances. This shows that optimizing the parameters is really important in order to obtain good performance.

Results are similar as in the case of our noisy sphere function. $best_{100}$ reaches the best score, but converges slowly. $mean_{100}$ has the fastest convergence, but finds the worst final score. As expected, $kbest_{100}$ is a compromise between the two previous fitness, with a nice convergence speed and is able to find a score really close to the best. For this reason, we have chosen to use *kbest* for the other problems.

**Results on all the problems.** We launched the optimization algorithm on all the problems from the set in the paper from Potvin and Bengio [11]. We compare the best score we obtained on each problem with our algorithm and the current best known score from the literature. The results are presented in table 2. We provide the Relative Percentage Deviation (RPD): $100 * (value - bestknown)/bestknown$.

| Problem | n | State of the art | Our best score | RPD | Problem | n | State of the art | Our best score | RPD |
|---|---|---|---|---|---|---|---|---|---|
| rc201.1 | 20 | 444.54 | **444.54** | 0 | rc205.1 | 14 | 343.21 | **343.21** | 0 |
| rc201.2 | 26 | 711.54 | **711.54** | 0 | rc205.2 | 27 | 755.93 | **755.93** | 0 |
| rc201.3 | 32 | 790.61 | **790.61** | 0 | rc205.3 | 35 | 825.06 | 828.27 | 0.39 |
| rc201.4 | 26 | 793.64 | **793.64** | 0 | rc205.4 | 28 | 760.47 | **760.47** | 0 |
| rc202.1 | 33 | 771.78 | 776.47 | 0.61 | rc206.1 | 4 | 117.85 | **117.85** | 0 |
| rc202.2 | 14 | 304.14 | **304.14** | 0 | rc206.2 | 37 | 828.06 | 839.18 | 1.34 |
| rc202.3 | 29 | 837.72 | **837.72** | 0 | rc206.3 | 25 | 574.42 | **574.42** | 0 |
| rc202.4 | 28 | 793.03 | **793.03** | 0 | rc206.4 | 38 | 831.67 | 859.07 | 3.29 |
| rc203.1 | 19 | 453.48 | **453.48** | 0 | rc207.1 | 34 | 732.68 | 743.29 | 1.45 |
| rc203.2 | 33 | 784.16 | **784.16** | 0 | rc207.2 | 31 | 701.25 | 707.74 | 0.93 |
| rc203.3 | 37 | 817.53 | 837.72 | 2.47 | rc207.3 | 33 | 682.40 | 687.58 | 0.76 |
| rc203.4 | 15 | 314.29 | **314.29** | 0 | rc207.4 | 6 | 119.64 | **119.64** | 0 |
| rc204.1 | 46 | 868.76 | 899.79 | 3.57 | rc208.1 | 38 | 789.25 | 797.89 | 1.09 |
| rc204.2 | 33 | 662.16 | 675.33 | 1.99 | rc208.2 | 29 | 533.78 | 536.04 | 0.42 |
| rc204.3 | 24 | 455.03 | **455.03** | 0 | rc208.3 | 36 | 634.44 | 641.17 | 1.06 |

**Table 2.** Results on all problems from the set from Potvin and Bengio [11]. First Row is the problem, second column the number of nodes, third column the best score found in [9], forth column the best score found by algorithm and fifth column it the RPD. The problems where we find the best solutions are in bold. We can see that for almost all problems with our simple algorithm we can find the best score.

There is a lot of differences between one set of parameters optimized on one problem and one set of parameters optimized on an other problem. So, the optimization has to be done on each problem.

We obtain as well as the state of the art for all the problems with less than 29 nodes. We find at least one correct solution for each problem. When the number of nodes increases, this is not a trivial task. For problems more difficult with a higher number of nodes, we don't do as well as the best score. However, we still manage to find a solution close to the current best one and did this with little domain knowledge.

## 5 Conclusion

In this paper we used a new method for solving the TSPTW problem based on the optimization of a nested Monte-Carlo algorithm with SA. This algorithm is a generic algorithm, used in many different applications. The only adaptation to the TSPTW was to add 3 heuristics. Even in this case, for all the problems with less than 29 nodes, we were able to reach state of the art solutions with small computation times. However, a clear limitation of our algorithm is dealing with a large number of nodes. A solution could be to prune some moves at the higher level of NMC. Other further work will be to add new heuristics. In this case, because of the higher dimensionality, we will try other evolution algorithms and increase the population size. A natural choice is the Covariance Matrix Self-Adaptation [3], known to be robust and good for large population sizes. Adding covariance and allowing large population sizes should increase the speed of the convergence.

## References

1. E. Baker. An exact algorithm for the time-constrained traveling salesman problem. *Operations Research*, 31(5):938–945, 1983.
2. H.-G. Beyer. *The Theory of Evolution Strategies.* Natural Computing Series. Springer, Heidelberg, 2001.
3. H.-G. Beyer and B. Sendhoff. Covariance matrix adaptation revisited - the CMSA evolution strategy. In G. Rudolph, T. Jansen, S. M. Lucas, C. Poloni, and N. Beume, editors, *Proceedings of PPSN*, pages 123–132, 2008.
4. T. Cazenave. Nested Monte-Carlo search. In *IJCAI*, pages 456–461, 2009.
5. N. Christofides, A. Mingozzi, and P. Toth. State-space relaxation procedures for the computation of bounds to routing problems. *Networks*, 11(2):145–164, 1981.
6. Y. Dumas, J. Desrosiers, E. Gelinas, and M. Solomon. An optimal algorithm for the traveling salesman problem with time windows. *Operations Research*, 43(2):367–371, 1995.
7. F. Focacci, A. Lodi, and M. Milano. A hybrid exact algorithm for the TSPTW. *INFORMS Journal on Computing*, 14(4):403–417, 2002.
8. M. Gendreau, A. Hertz, G. Laporte, and M. Stan. A generalized insertion heuristic for the traveling salesman problem with time windows. *Operations Research*, 46(3):330–335, 1998.
9. M. López-Ibáñez and C. Blum. Beam-ACO for the travelling salesman problem with time windows. *Computers & OR*, 37(9):1570–1583, 2010.
10. G. Pesant, M. Gendreau, J. Potvin, and J. Rousseau. An exact constraint logic programming algorithm for the traveling salesman problem with time windows. *Transportation Science*, 32(1):12–29, 1998.
11. J. Potvin and S. Bengio. The vehicle routing problem with time windows part II: genetic search. *INFORMS journal on Computing*, 8(2):165, 1996.
12. I. Rechenberg. *Evolutionstrategie: Optimierung Technischer Systeme nach Prinzipien der Biologischen Evolution.* Fromman-Holzboog Verlag, Stuttgart, 1973.
13. M. Savelsbergh. Local search in routing problems with time windows. *Annals of Operations Research*, 4(1):285–305, 1985.
14. H.-P. Schwefel. Adaptive Mechanismen in der biologischen Evolution und ihr Einfluss auf die Evolutionsgeschwindigkeit. Interner Bericht der Arbeitsgruppe Bionik und Evolutionstechnik am Institut für Mess- und Regelungstechnik Re 215/3, Technische Universität Berlin, Juli 1974.
15. M. Solomon. Algorithms for the vehicle routing and scheduling problems with time window constraints. *Operations Research*, 35(2):254–265, 1987.
16. F. Teytaud. A new selection ratio for large population sizes. *Applications of Evolutionary Computation*, pages 452–460, 2010.