Beam Nested Rollout Policy Adaptation

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

[1] LAMSADE, Université Paris Dauphine, France,
[2] HEC Paris, CNRS, 1 rue de la Libération 78351 Jouy-en-Josas, France

Abstract. The Nested Rollout Policy Adaptation algorithm is a tree search algorithm known to be efficient on combinatorial problems. However, one problem of this algorithm is that it can converge to a local optimum and get stuck in it. We propose a modification which limits this behavior and we experiment it on two combinatorial problems for which the Nested Rollout Policy Adaption is known to be good at.

1 Introduction

Recently, the Nested Monte-Carlo Search (NMC) has been proposed for solving combinatorial problems [2, 13]. Based on this algorithm, a new algorithm has been successfully introduced, the Nested Rollout Policy Adaptation algorithm [14]. This algorithm is efficient for numerous combinatorial problems, and in particular, the Traveling Salesman Problem with time Windows [3] and the Morpion Solitaire puzzle [14].
The idea behind the NMC algorithm can been seen as a Meta Monte-Carlo algorithm. This is a recursive algorithm. The first level of the search consists in simply performing a Monte-Carlo simulation, i.e. each decision is chosen randomly until no more possible decision are available. At the end, the score of the position that has been reached is sent back. This first level is called the level 0. For each other level $lvl > 0$, the search consists in launching a NMC algorithm with a level $lvl - 1$ for each possible decision. Such as for the level 0, the score for each reached position is sent back, and the decision with the best score is chosen. This algorithm is presented in Section 3. The NRPA algorithm is based on this idea, except that a level 0 policy is learned by gradient ascent and is used instead of the Monte-Carlo policy. This algorithm is presented in Section 4. One problem of this algorithm is that it can converge to local optima due to its simple learning. In this work, we propose a modification in order to improve the behavior of the NRPA algorithm in front of local optima. The principle of the modification consists in keeping a beam of different sequences (with their corresponding policies).

The paper is organized as follows. The next section (Section 2) presents the two problems studied in this work, the Traveling Salesman Problem with Time Windows in Section 2.1 and the Morpion-Solitaire puzzle in Section 2.2. In Section 3, the Nested Monte-Carlo Search is presented, in Section 4 we present the Nested Rollout Policy Adaptation algorithm, and in Section 5 the improvement done on the NRPA algorithm. Finally, in Section 6 we present comparisons between the NRPA algorithm and the algorithm designed in this work.

2 Problems

In this Section we present two well-known combinatorial problems. The first one, presented in Section 2.1 is the Traveling Salesman Problem with Time Windows. The second problem is the puzzle called Morpion-Solitaire and is presented in Section 2.2. The NRPA algorithm has been already used for solving these two problems [14, 3].

2.1 The Traveling Salesman Problem with Time Windows

The Traveling Salesman Problem (TSP) is a well-known logistic problem. Given a list of cities and their pairwise distances, the goal is to find the shortest possible path that visits each city only once. The path has to start and finish at a given depot. The TSP problem is NP-hard [9]. The Traveling Salesman Problem with Time Windows (TSPTW) is a problem based on the TSP. Inputs are the same, but a difficulty is added. In this version, a time interval is defined for each city, and each city has to be visited within its corresponding period of time.

Formally, the TSPTW can be defined as follows. Let G be an undirected complete graph. $G = (N, A)$, where $N = 0, 1, \ldots, n$ corresponds to a set of nodes and $A = N \times N$ corresponds to the set of edges between the nodes. The node 0 corresponds to the depot. Each city is represented by the n other nodes. A cost function $c : A \to \mathbb{R}$ is given and represents the distance between two cities. A solution to this problem is a sequence of nodes $P = (p_0, p_1, \ldots, p_n)$ where $p_0 = 0$ and $(p_1, \ldots, p_n)$ is a permutation of $[1, N]$. Set $p_{n+1} = 0$ (the path must finish at the depot), then the goal is to minimize the function defined in Equation 1.

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

As said previously, the TSPTW version is more difficult because each city i has to be visited in a time interval $[e_i, l_i]$. This means that a city i has to be visited before l_i. It is possible to visit a city before e_i, but in that case, the new departure time becomes e_i. Consequently, this case may be dangerous as it generates a penalty. Formally, if r_{p_k} is the real arrival time at node p_k, then the departure time d_{p_k} from this node is $d_{p_k} = max(r_{p_k}, e_{p_k})$.

In the TSPTW, the function to minimize is the same as for the TSP (Equation 1), but a set of constraints is added and must be satisfied. Let us define $\Omega(P)$ as the number of violated windows constraints by tour (P).
Two constraints are defined. The first constraint is to check that the arrival time is lower than the fixed time. Formally,

$$\forall p_k, r_{p_k} < l_{p_k}.$$

The second constraint is the minimization of the time lost by waiting at a city. Formally,

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

With algorithms used in this work, paths with violated constraints can be generated. As presented in [13], a new score $Tcost(p)$ of a path p can be defined as follows:

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

with, as defined previously, $cost(p)$ the cost of the path p and $\Omega(p)$ the number of violated constraints. 10^6 is a constant chosen high enough so that the algorithm first optimizes the constraints.

A survey of efficient methods for solving the TSPTW can be found in [10]. Existing methods for solving the TSPTW are numerous. First, branch and bound methods were used [1, 4]. Later, dynamic programing based methods [6], heuristics based algorithms [15, 8] and methods based on constraint programming [7, 11] have been published. More recently, ant colony optimization algorithms have been used [10] and have established new state of the art scores. Works based on the NMC have been proposed in [13] and on the NRPA in [3].

2.2 Morpion-Solitaire

Morpion-Solitaire is an NP-hard pencil-and-paper puzzle played on a square grid. A move consists in adding a circle (on one possible intersection on the grid) such that a line containing five circles can be drawn. The new line is then added to the grid. Lines can be horizontal, vertical or diagonal. The initial grid contains some starting circles, as shown in Figure 1. Two versions of this puzzle exist, the touching version and the disjoint version. In this paper, we are interested in the first one, the disjoint version, for which a circle can not belong to two lines that have the same direction. The best human score for this version of the puzzle is 68 moves [5]. The Nested Monte-Carlo search found a score of 80 moves [2], and [14] found a new record with 82 moves.

3 Nested Monte-Carlo Search

The basic idea of Nested Monte-Carlo Search is to perform a principal playout with a bias on the selection of each decision based on the results of a Monte-Carlo tree search [2].

The base level of the search build random solutions (i.e. playouts), random decision are chosen until the end at this level. When a solution is completely built, the score of the position that has been reached is sent back.

At each decision of a playout of level 1 it chooses the decision that gives the best score when followed by a random playout. Similarly for a playout of level n it chooses the decision that gives the best score when followed by a playout of level $n - 1$.

When a search at the highest level is finished and there is time left, another search is performed at the highest level, and so on until the thinking time is elapsed.

Nested Monte-Carlo search has been successful in establishing world records in single player games such as Morpion Solitaire or SameGame [2]. It provides a good balance between exploration and exploitation and it automatically adapts its search behavior to the problem at hand without parameters tuning.

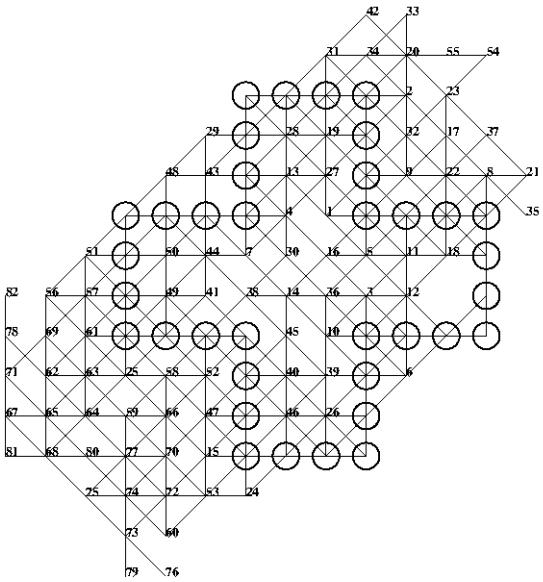

Fig. 1. Example of a puzzle. Circles represent initial points and numbers represent the moves. This 82 moves grid found by our algorithm equalizes the world record established by Rosin [14] through a different solution.

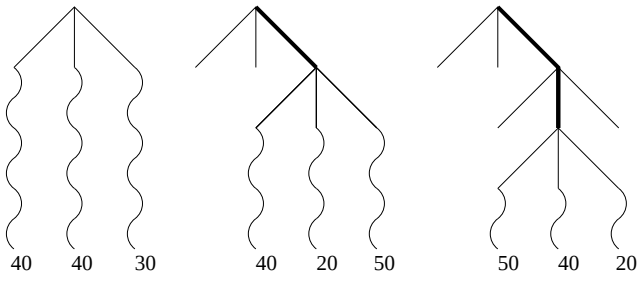

Fig. 2. At each step of the principal playout shown here with a bold line, an NMC of level n performs a NMC of level $n-1$ (shown with wavy lines) for each available decision and selects the best one. At level 0, a simple pseudo-random playout is used.

Figure 2 illustrates a level 1 Nested Monte-Carlo search. Three selections of cities at level 1 are shown. The leftmost tree shows that, at the root, all possible cities are tried and that for each possible decision a playout follows it. Among the three possible cities at the root, the rightmost city has the best result of 30, therefore this is the first decision played at level 1. This brings us to the middle tree. After this first city choice, playouts are performed again for each possible city following the first choice. One of the cities has result 20 which is the best playout result among his siblings. So the algorithm continues with this decision as shown in the rightmost tree. This algorithm is presented in Algorithm 1.

Algorithm 1 Nested Monte-Carlo search

```
nested (level,node)
if level==0 then
  ply ← 0
  seq ← {}
  while num_children(node) > 0 do
    CHOOSE seq[ply] ← child i with probability 1/num_children(node)
    node ← child(node,seq[ply])
    ply ← ply+1
  end while
  RETURN (score(node),seq)
else
  ply ← 0
  seq ←{}
  best_score ← ∞
  while num_children(node) > 0 do
    for children i of node do
      temp ← child(node,i)
      (results,new) ← nested(level-1,temp)
      if results<best_score then
        best_score ← results
        seq[ply]=i
        seq[ply+1...]=new
      end if
    end for
    node=child(node,seq[ply])
    ply ← ply+1
  end while
  RETURN (best_score,seq)
end if
```

At each choice of a playout of level 1 it chooses the city that gives the best score when followed by a single random playout. Similarly for a playout of level n it chooses the city that gives the best score when followed by a playout of level $n-1$.

4 The Nested Rollout Policy Adaptation algorithm

The Nested Rollout Policy Adaptation algorithm (NRPA) is an algorithm that learns a playout policy. There are different levels in the algorithm. Each level is associated to the best sequence found at that level. The playout policy is a vector of weights that are used to calculate the probability of choosing a city. A city is chosen proportionally to the exponential of its associated weight. Learning the playout policy consists in increasing the weights associated to the best cities and decreasing the weights associated to the other cities. The algorithm is given in Algorithm 2.

Algorithm 2 Nested Rollout Policy Adaptation

```
NRPA (level,pol)
if level = 0 then
  node ← root
  ply ← 0
  seq ← {}
  while there are possible decisions do
    CHOOSE seq[ply] ← child i the with probability proportional to exp(pol[code(node,i)])
    node ← child(node, seq [ply])
    ply ← ply + 1
  end while
  return (score (node), seq)
else
  bestScore ← ∞
  for N iterations do
    (result,new) ← NRPA (level − 1, pol)
    if result ≤ bestScore then
      bestScore ← result
      seq ← new
    end if
    pol ← Adapt(pol,seq)
  end for
end if
return (bestScore,seq)

Adapt (pol,seq)
node ← root
pol′ ← pol
for ply ← 0 to length(seq) - 1 do
  pol′[code(node,seq[ply])] += Alpha
  z ← SUM exp(pol[code(node,i)]) over node's children i
  for children i of node do
    pol′[code(node,i)] -= Alpha × exp(pol[code(node,i)]) / z
  end for
  node ← child(node, seq [ply])
end for
return pol′
```

5 The Beam Nested Rollout Policy Adaptation algorithm

The idea of Beam Nested Rollout Policy Adaptation is to combine a beam search with the Nested Rollout Policy Adaptation algorithm. Instead of memorizing one sequence at each level of the algorithm, a set of the best sequences is memorized at each level. The size of the beam for a given level is the number of sequences in the set of this level. Note that the sequences are not memorized alone. Each memorized sequence is associated to a score and a policy. The algorithm is given in Algorithm 3. In the algorithm r is a score, s is a sequence and p is a policy.

As can be seen in the algorithm, a recursive call is performed for each sequence in the set of best sequences for each level. At the end of the algorithm a set of the best sequences and the associated policies and scores is returned. This set is used to adapt the policies at the upper level and these adapted policies are inserted in the set of best sequences at the upper level. When all the sequences coming from the calls at the lower level have been inserted, only the B best ones are kept (B being the size of the beam at that level).

The Adapt function that learns the policy is the same as in the original NRPA algorithm.

Algorithm 3 Beam Nested Rollout Policy Adaptation

```
beamNRPA (level,pol)
if level = 0 then
  node ← root
  ply ← 0
  seq ← {}
  while there are possible decisions do
    CHOOSE seq[ply] ← child i the with probability proportional to exp(pol[code(node,i)])
    node ← child(node, seq [ply])
    ply ← ply + 1
  end while
  return (score (node), seq, pol)
else
  beam ← {(∞,{},pol)}
  for N iterations do
    newBeam ← {}
    for (r,s,p) in beam do
      insert (r,s,p) in newBeam
      beam1 ← beamNRPA (level − 1,p)
      for (r1, s1, p1) in beam1 do
        p1 ← Adapt(p,s1)
        insert (r1,s1,p1) in newBeam
      end for
    end for
    beam ← B best scores of newBeam
  end for
  return beam
end if
```

6 Experimental Results

We apply the beam NRPA algorithm to two applications, the TSPTW, presented in Section 2.1 and the Morpion-Solitaire puzzle, presented in Section 2.2. Results are presented respectively in Section 6.1 and in Section 6.2. We define the complexity of the algorithm as the total number of evaluations (rollout) done by the algorithm. Formally, for the beam NRPA algorithm, the complexity is

$$C = (N * B)^{lvl}$$

with B the size of the beam, lvl the level of the algorithm and N the number of iterations done for the learning. Experimentally, we have found that having a beam size $B > 1$ only for the level 1 was the best choice in terms of complexity. The complexity becomes then

$$C = N^{lvl} * B$$

. For all our experiments, the size of the beam is fixed to 1 for all levels above 1 and is changed at level 1. Consequently, increasing the complexity comes to increase the size of the beam at level 1. In order to have comparable complexities for both beam NRPA and NRPA algorithms, we repeat the NRPA algorithm B times, and we take the best value found during the B runs as the return value of the algorithm.

6.1 Traveling Salesman Problem with Time Windows

In a first experiment on the TSPTW, we compare the best score found by the two algorithms on two fixed problems from the set of problems from [12]. The two problems are the problem rc203.1, which is a simple one, with 19 cities, and the rc202.3, which has 29 cities and is then harder. We measure the average traveling score as a function of C. We experiment $N = \{20, 50, 100\}$ and $B = \{2, 4, 8, 16, 32, 64\}$ for $N = 20$, $B = \{2, 4, 8, 16\}$ for $N = 50$ and $B = \{2, 4, 8\}$ for $N = 100$. Results for the problem rc203.1 are presented in Figure 3. We experiment three different values of N in level 2. The beam NRPA is always better than the classic algorithm for all complexities (i.e., for all different sizes of beam). $N = 20$ and $N = 50$ for the beam algorithm are the only versions that are able to find valid paths (i.e. without violated constraints).

For the second problem (rc202.3), results are presented in Figure 4. Here again, it is always better to use the beam NRPA algorithm. We can note that, because this problem is harder, a larger value of N is needed, meaning that more time need to be spent during the learning phase. Best results are found with the beam NRPA algorithm with $N = 50$ and $N = 100$.

The last experiment on the TSPTW, is to run the beam NRPA algorithm on all problems from the set of problems from [12], and to compare our results with the results found by the NRPA algorithm from [3]. Results are presented in Table 1.

As expected, we can see that the beam NRPA algorithm is always able to find better scores than the NRPA algorithm. The beam NRPA is able to find 63% of state of the art scores, and this without any expert knowledge. Expert knowledge can be added to the beam NRPA algorithm, in the same way as in the NRPA version from [3].

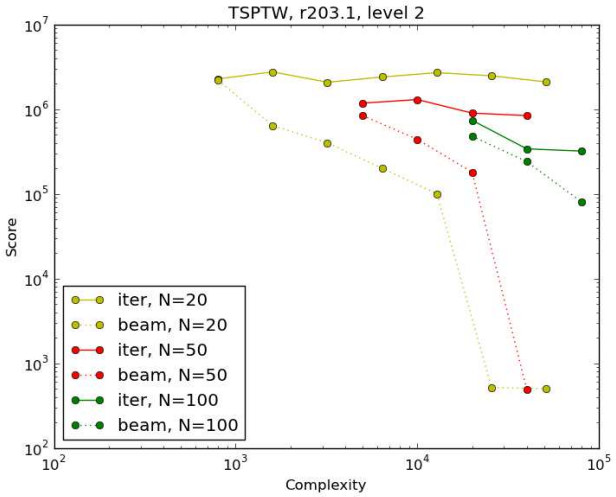

Fig. 3. Experience on the problem rc203.1 with level 2. The lower the better. Average on 30 runs. Best results are found by the beam NRPA algorithm with $N = 20$. In this experiment, only the beam NRPA algorithm is able to find a valid path, without violated constraints. The best known score for this problem is 453.48. This score is reached for the Beam NRPA with $N = 20$.

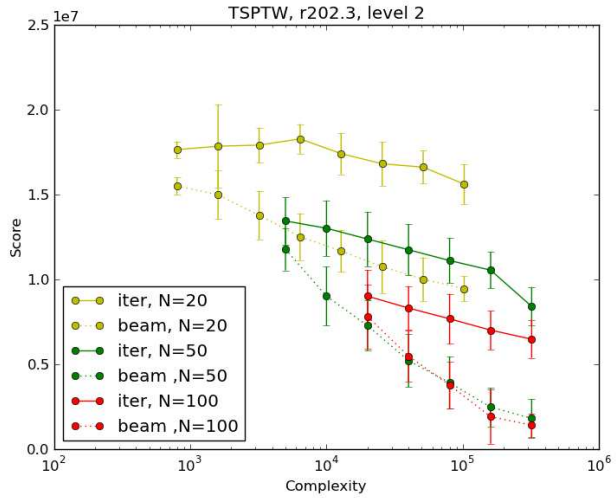

Fig. 4. Experience on the problem rc202.3 with level 2. The lower the better. Average on 30 runs. Best results are found by the beam NRPA algorithm with $N = 50$ and $N = 100$. The best known score for this problem is 837.72

Problem	City	State of the art	NRPA	beam NRPA
rc206.1	4	117.85	**117.85**	**117.85**
rc207.4	6	119.64	**119.64**	**119.64**
rc202.2	14	304.14	**304.14**	**304.14**
rc205.1	14	343.21	**343.21**	**343.21**
rc203.4	15	314.29	**314.29**	**314.29**
rc203.1	19	453.48	**453.48**	**453.48**
rc201.1	20	444.54	**444.54**	**444.54**
rc204.3	24	455.03	**455.03**	**455.03**
rc206.3	25	574.42	**574.42**	**574.42**
rc201.2	26	711.54	**711.54**	**711.54**
rc201.4	26	793.64	**793.64**	**793.64**
rc205.2	27	755.93	**755.93**	**755.93**
rc202.4	28	793.03	800.18	**793.03**
rc205.4	28	760.47	765.38	765.38
rc202.3	29	837.72	839.58	839.58
rc208.2	29	533.78	537.74	**533.78**
rc207.2	31	701.25	702.17	702.17
rc201.3	32	790.61	796.98	795.43
rc204.2	33	662.16	673.89	663.19
rc202.1	33	771.78	775.59	772.17
rc203.2	33	784.16	**784.16**	798.73
rc207.3	33	682.40	688.50	**682.40**
rc207.1	34	732.68	743.72	**732.68**
rc205.3	35	825.06	828.36	**825.06**
rc208.3	36	634.44	656.40	649.93
rc203.3	37	817.53	820.93	**817.53**
rc206.2	37	828.06	829.07	842.17
rc206.4	38	831.67	831.72	**831.67**
rc208.1	38	789.25	799.24	795.57
rc204.1	46	868.76	883.85	878.76

Table 1. Results on all problems from the set from Potvin and Bengio [12]. First Column corresponds to the problem, second column is the number of cities, third column is the state of the art score, found in [10]. Fourth column is the best score found by the NRPA algorithm in [3] and fifth column is the best score found by the beam NRPA algorithm. The problems for which we find the state of the art solutions are in bold. With the beam NRPA 63% of state of the art scores are found, where as with the classic NRPA algorithm only 43% state of the art scores are found.

6.2 Morpion-Solitaire

The second experimented application is the Morpion-Solitaire puzzle. As for the two first experiments on the TSPTW, we measure the best score found by the beam NRPA and the NRPA algorithms as a function of the complexity in level 2. For this application, the higher scores the better. Results are presented in Figure 5. For beam sizes larger than 2, results are always better for the beam NRPA algorithm. For $B = 2$, results are equivalent.

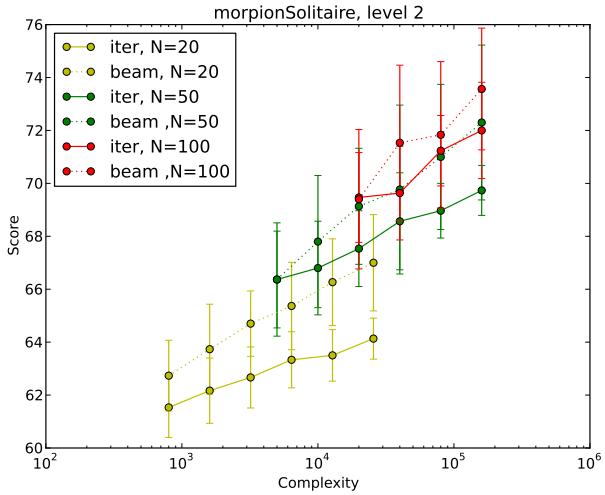

Fig. 5. Experience on the Morpion-Solitaire puzzle with level 2. The higher the better. Best results are found by the beam NRPA algorithm with $N = 100$. Each point is an average of 30 runs.

7 Conclusion

In this work we show how to improve the Nested Rollout Policy Adaptation algorithm. For both applications, results are good for a beam size of 4. When the size of the beam increases, results are even better. On the first experimented application, the traveling salesman problem with time windows, we do not use any expert knowledge. Our goal was then, not to find new records, but to show the efficiency of having numerous learned policies. The classic NRPA algorithm find 43% of state of the art records, whereas the beam NRPA algorithm is able to find 63% of records. Only for 2 problems we are not able to find equal or better scores than the NRPA algorithms. For all other problems, scores are equal or better for the beam NRPA algorithm. On the Morpion Solitaire puzzle, we reach the current record (82 moves), but we are not able to beat it. However, as shown in Figure 5 best scores are found faster with the beam algorithm than with

the classic NRPA algorithm. This behaviour has been also observed for the traveling salesman problem with time windows (Figures 4 and 3).
As pointed out in the future works of the NRPA algorithm's author in [14], realizing a parallel version of the NRPA algorithm is a challenging work. The beam NRPA algorithm has the advantage to be easily parallelizable, because, all policies from the beam can be evaluated in parallel.
An interesting future work is to keep distances between all the sequences from the beam. Having such a modification should be much more robust in front of local optima.

References

1. E.K. Baker, 'An exact algorithm for the time-constrained traveling salesman problem', *Operations Research*, **31**(5), 938–945, (1983).
2. T. Cazenave, 'Nested Monte-Carlo search', in *IJCAI*, pp. 456–461, (2009).
3. T. Cazenave and F. Teytaud, 'Application of the nested rollout policy adaptation algorithm to the traveling salesman problem with time windows', in *LION 6*. Springer, (2012).
4. 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).
5. E.D. Demaine, M.L. Demaine, A. Langerman, and S. Langerman, 'Morpion solitaire', *Theory of Computing Systems*, **39**(3), 439–453, (2006).
6. Y. Dumas, J. Desrosiers, E. Gelinas, and M.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. D.S. Johnson and C.H. Papadimitriou, *Computational complexity and the traveling salesman problem*, Mass. Inst. of Technology, Laboratory for Computer Science, 1981.
10. Manuel López-Ibáñez and Christian Blum, 'Beam-ACO for the travelling salesman problem with time windows', *Computers & OR*, **37**(9), 1570–1583, (2010).
11. G. Pesant, M. Gendreau, J.Y. Potvin, and J.M. Rousseau, 'An exact constraint logic programming algorithm for the traveling salesman problem with time windows', *Transportation Science*, **32**(1), 12–29, (1998).
12. J.Y. Potvin and S. Bengio, 'The vehicle routing problem with time windows part II: genetic search', *INFORMS journal on Computing*, **8**(2), 165, (1996).
13. A. Rimmel, F. Teytaud, and T. Cazenave, 'Optimization of the nested monte-carlo algorithm on the traveling salesman problem with time windows', *Applications of Evolutionary Computation*, 501–510, (2011).
14. Christopher D. Rosin, 'Nested rollout policy adaptation for monte carlo tree search', in *IJCAI*, pp. 649–654, (2011).
15. M.M. Solomon, 'Algorithms for the vehicle routing and scheduling problems with time window constraints', *Operations Research*, **35**(2), 254–265, (1987).