Monte Carlo Search Algorithms for Network Traffic Engineering

Chen Dang[1,2(✉)], Cristina Bazgan[2], Tristan Cazenave[2], Morgan Chopin[1], and Pierre-Henri Wuillemin[3]

[1] Orange Labs, Châtillon, France
{chen.dang,morgan.chopin}@orange.com
[2] Université Paris-Dauphine, PSL Research University, CNRS, UMR 7243, LAMSADE, 75016 Paris, France
{cristina.bazgan,tristan.cazenave}@dauphine.psl.eu
[3] Sorbonne Université, CNRS, UMR 7606, LIP6, 75005 Paris, France
pierre-henri.wuillemin@lip6.fr

Abstract. The aim of Traffic Engineering is to provide routing configurations in networks such that the used resources are minimized while maintaining a high level of quality of service (QoS). Among the optimization problems arising in this domain, we address in this paper the one related to setting weights in networks that are based on shortest path routing protocols (OSPF, IS-IS). Finding weights that induce efficient routing paths (e.g. that minimize the maximum congested link) is a computationally hard problem.

We propose to use Monte Carlo Search for the first time for this problem. More specifically we apply Nested Rollout Policy Adaptation (NRPA). We also extend NRPA with the *force_exploration* algorithm to improve the results. In comparison to other algorithms NRPA scales better with the size of the instance and can be easily extended to take into account additional constraints (cost utilization, delay, ...) or linear/non-linear optimization criteria. For difficult instances the optimum is not known but a lower bound can be computed. NRPA gives results close to the lower bound on a standard dataset of telecommunication networks.

Keywords: Traffic engineering · Policy adaptation · Monte Carlo search

1 Introduction

Despite the emergence of new network routing technologies such as Segment Routing or MPLS (*MultiProtocol Label Switching*), many telecommunication networks still mostly rely on the computation of shortest paths for the transportation of packets, such as Open Shortest Path First (OSPF) or Intermediate System to Intermediate System (IS-IS). In such routing protocols, the network manager controls the data flow by simply supplying so-called administrative weights to the links of the networks. Then, every packet is routed from its origin

Y. Dong et al. (Eds.): ECML PKDD 2021, LNAI 12978, pp. 486–501, 2021.
https://doi.org/10.1007/978-3-030-86514-6_30

to its destination along the shortest paths induced by those weights. While this method has the advantage of being easy to manage, it lacks precise control over the paths that are elected to route the traffic because one can only modify those paths indirectly by changing the weights. As a consequence, the main challenge for a network manager is to find a set of weights that induce routing paths such that the load is minimized while maintaining a high level of QoS on operational networks. Unfortunately, this task turns out to be computationally hard to solve. In this paper, we are interested in one of the network optimization problems related to this issue: Given a bidirected graph and a set of demands, the task is to find a set of weights such that the demands routed along the induced shortest paths generate a minimum congestion *i.e.,* the maximum ratio value of the total traffic going through an edge over the edge's capacity is minimized. There are mainly two variants of this optimization problem studied in the literature namely the *splittable* and *unsplittable* versions. In the former, we allow each demand to be routed along several shortest paths while, in the latter, each demand is required to be routed on a unique shortest path between its origin and its destination.

Several authors proposed to solve this problem using integer programming models and meta-heuristics methods, the reader is referred to [6] for a complete overview of these approaches. Regarding the splittable variant of the problem, Fortz and Thorup [23] showed that it is NP-hard to approximate within a factor $\frac{3}{2} - \epsilon$ for all $\epsilon > 0$. Hence, to cope with the hardness of this problem, many different meta-heuristics approaches were investigated. Fortz and Thorup [23] first proposed a local search algorithm to solve the splittable variant which was latter implemented in the TOTEM library [27] and called IGP-WO. This approach was further extended to compute robust solutions against single link failures [24] or in the context of oblivious routing [2]. Genetic algorithms were also proposed to solve this problem [8,21]. The reader is referred to the surveys [1,22] for more details and references about the existing meta-heuristics approaches proposed to solve this splittable variant. Most of the previous meta-heuristics were tested on networks of small/moderate size and do not consider the unsplittable case or QoS constraints such as the delay. Regarding the unsplittable case, Bley [4] showed that this variant is NP-hard even on bidirected cycles and not $O(n^{1-o(1)})$-approximable unless $P = NP$ in the general case. In [5], the author proposed an exact algorithm using a two-phase approach: the problem is decomposed into a master problem that aims at finding an optimal shortest path routing, and a client problem which consists in finding a compatible set of weights for those shortest paths. This master problem is modeled using an integer linear program and solved using a branch-and-cut algorithm. In [3], further exact algorithms are proposed either based on a compact formulation of the problem or a dynamic programming algorithm using a tree decomposition of the input graph. Unfortunately, the current exact methods can only handle networks of moderate size (e.g. dozens of nodes) while real networks can have hundreds of routers and links. In this paper, we propose to use a Monte Carlo Search approach in order to get algorithms that (*i*) achieve a better scalability and (*ii*) can easily be extended to integrate operational constraints (unique shortest paths, delay, ...).

Monte Carlo Search algorithms have been successfully applied to many difficult problems but not yet to telecommunication networks optimization. We address in this paper the use of Monte-Carlo Search for this difficult problem. We compare UCT [26], Nested Monte Carlo Search (NMCS) [9] and Nested Rollout Policy Adaptation (NRPA) [32] which is an algorithm that learns a playout policy online on each instance. NMCS is an algorithm that works well for puzzles. It biases its playouts using lower level playouts. At level zero NMCS adopts a uniform random playout policy. Online learning of playout strategies combined with NMCS has given good results on optimization problems[31]. Other applications of NMCS include Single Player General Game Playing [28], Coding Theory [25], Cooperative Pathfinding [7], Software testing and heuristic Model-Checking [30], the Pancake problem, Games [13], Cryptography and the RNA inverse folding problem.

Online learning of a playout policy in the context of nested searches has been further developed for puzzles and optimization with Nested Rollout Policy Adaptation (NRPA) [32]. NRPA has found new world records in Morpion Solitaire and crosswords puzzles. Edelkamp, Cazenave and co-workers have applied the NRPA algorithm to multiple problems. They have optimized the algorithm for the Traveling Salesman with Time Windows (TSPTW) problem [15,16]. Other applications deal with 3D Packing with Object Orientation [18], the physical traveling salesman problem [19], the Multiple Sequence Alignment problem [20], Logistics [11,17], Graph Coloring [12] and Inverse Folding [10]. The principle of NRPA is to adapt the playout policy so as to learn the best sequence of moves found so far at each level.

The paper is organized as follows. Section 2 is devoted to the basic definitions and presentation of the optimization problem. Section 3 explains the application of Monte Carlo Search to the routing problem. Section 4 gives experimental results while concluding remarks and future research directions are given in Sect. 5.

2 Problem Formulation

In this paper we consider a bidirected graph that is a digraph where, for any arc uv, the reverse arc vu is also present. Given a bidirected graph $G = (V, A)$, every vertex $v \in V$ corresponds to a router while an arc uv corresponds to a link between routers u and v. Every arc uv is associated a capacity denoted by c_{uv}. Let K denote a set of demands or commodities to be routed in G. Each demand $k \in K$ is defined by a pair of vertices s^k and t^k representing the source and the target of k, a traffic volume D^k to be routed from s^k to t^k. Such a demand k will be denoted by the quadruplet (s^k, t^k, D^k). Given a metric $w \in Z_+^{|A|}$, each demand $k \in K$ is routed along the shortest paths between s^k and t^k. If there are more than one shortest paths joining the extremities of k, the traffic volume D^k is splitted evenly among those paths according to the so-called ECMP (Equal-Cost Multi-Path) rule. More precisely, the traffic volume that reaches a node $v \in V$ must be split equally among all arcs leaving v and belonging to the shortest

paths toward destination t^k. We then define the *load* of an arc uv induced by w, denoted by $load(uv, w)$, as the amount of traffic traversing the arc uv over its capacity (see Fig. 1). The *congestion* $cong(w)$ of a given metric w is defined by $\max_{uv \in A} load(uv, w)$, that is the maximum load over all arcs.

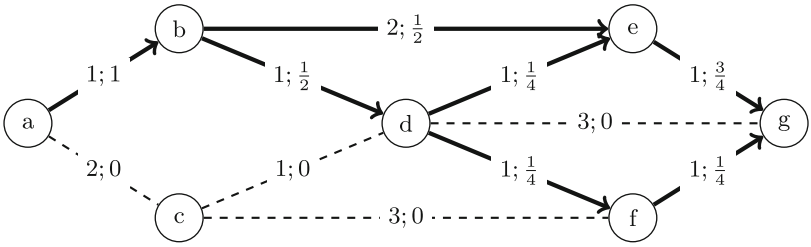


Fig. 1. Illustration of a shortest path routing with the ECMP rule. In this figure, we assume unit capacities and suppose that a demand k with traffic volume $D^k = 1$ must be routed from $s^k = a$ to $t^k = g$. A label $w_{uv}; load(uv, w)$ is associated to each arc $uv \in A$.

We are now in position to define the optimization problem studied in this paper. The Minimum Congestion Shortest Path Routing (Min-Con-SPR) problem is to find a metric $w \in Z_+^{|A|}$ and the routing paths induced by these weights such that the network congestion $cong(w)$ is minimum. The problem Min-Con-SPR can be defined formally as follows:

Min-Con-SPR

Input: A bidirected graph $G = (V, A)$, where each arc uv has a capacity $c_{uv} \geq 0$ and a set of communities K defined for each $k \in K$ by the quadruplet (s^k, t^k, D^k).

Output: A metric $w \in Z_+^{|A|}$ of minimum congestion $cong(w)$.

In this paper, we also consider the Min-Con-SPR problem with some or all of the following additional constraints

Unicity: In this constraint, we require that each demand is routed along a uniquely determined shortest path.

Delay: This constraint requires that the routing paths have length at most the length of a shortest (s^k, t^k)-path (in terms of number of arcs) plus a constant $c \in N^+$.

From an operational point of view, the unicity constraint is sometimes required by the network manager to monitor the flow circulating in the system more easily. In addition to minimizing the congestion, the delay constraint ensure a certain level of QoS regarding the latency of answering the requests made by the clients.

It is worth noting that all of our results regarding the delay constraint can easily be extended to the more general case where each arc is associated with a latency value and each demand k has a delay value Δ^k and must be routed along shortest paths with total latency value less than Δ^k.

3 Monte Carlo Search on Routing Problem

Monte Carlo Search is a general optimization technique. We detail in this section how it can be used and improved for Min-Con-SPR with or without the previous additional constraints.

3.1 Monte Carlo Search

In this section, we present three different Monte Carlo search-based approaches which are applicable to the target problem. The first approach is UCT (Upper Confidence Trees), which uses bandit ideas to guide Monte Carlo planning [26]. Assuming the state s, playouts will be completed in a certain amount of time and statistics about the states and the actions will be collected. Supposing the action space for state s is $\mathcal{A}(s)$, the action a is chosen such that the upper bound of the score is maximized:

$$a_s = \underset{a \in \mathcal{A}(s)}{arg\,max} \left(\bar{Q}_{s,a} + \tau \sqrt{\frac{\ln(N_s)}{N_{s,a}}} \right)$$

where $\bar{Q}_{s,a}$ is the estimated score of the action a at state s, N_s is the number of times state s was visited, $N_{s,a}$ is the number of times action a was selected at state s. τ is a constant value which controls the degree of exploration.

Another approach is NMCS (Nested Monte Carlo Search) [13]. By nesting the evaluation function inside another evaluation function, the ability of the traditional Monte Carlo is greatly improved. However this approach is more sensible to the size of the search space.

Algorithm 1: The playout algorithm

Function `playout`(*state, policy*)**:**
- *sequence* ← []
- **while** *state is not terminal* **do**
  - $z \leftarrow \sum_{a' \in \mathcal{A}(state)} e^{policy[code(state,a')]}$
  - Draw a with probability $\frac{1}{z} e^{policy[code(state,a)]}$
  - *state* ← play(*state*, a)
  - append a to *sequence*
- **end**
- **return** (score(*state*), *sequence*)

NRPA (Nested Rollout Policy Adaptation) [32] is also used in our study. The NRPA can be decomposed into three principal functions: the playout function, the adapt function and the NRPA function. The Algorithms 1 and 2 show the three functions respectively. The NRPA use a domain specific code $code(state, a)$ for the action a in the representation of the policy, where many actions may share the same code, and actions with different codes are searched separately. For each nesting level, NRPA recursively calls to the lower level, searching to improve its

current best score. When it succeeds, the best score of the corresponding state $score(state)$ is updated, and the current action sequence is recorded as the best sequence.

Algorithm 2: The adapt and NRPA algorithm

Function `adapt`(*policy, sequence,* α)**:**
  $pol \leftarrow policy$
  $state \leftarrow root$
  for *a in sequence* **do**
    $z \leftarrow \sum_{a' \in \mathcal{A}(state)} e^{policy[code(state,a')]}$
    $\forall a' \in \mathcal{A}(state), pol[code(state, a')] -= \alpha * \frac{1}{z} e^{policy[code(state,a')]}$
    $pol[code(state, a)] += \alpha$
    $state \leftarrow \text{play}(state, a)$
  end
  return *pol*
end
Function `NRPA`(*level, policy*)**:**
  if $level == 0$ **then**
    return playout(*root*, *policy*)
  end
  else
    $bestScore \leftarrow \text{inf}$
    for *N iterations* **do**
      $(result, new) \leftarrow \text{NRPA}(level - 1, policy)$
      if $result \leq bestScore$ **then**
        $bestScore \leftarrow result$
        $seq \leftarrow new$
      end
      $policy \leftarrow \text{adapt}(policy, seq)$
    end
    return (*bestScore*, *seq*)
  end
end

3.2 Modeling with Monte Carlo Search

To model the Min-Con-SPR problem with Monte Carlo Search algorithms, we suppose that a solution to the Min-Con-SPR problem is represented by a point (i.e. the metric $w = \langle w_1, w_2, ..., w_{|A|} \rangle$) in the discrete space $[1, 65535]^{|A|}$. To reduce the search space, we set the value space of the metric as a subspace W of the original space $[1, 65535]$.

For each playout, the metric of the graph is assigned and the objective function is evaluated. In our case, $cong(w)$ is used as the score. After obtaining the congestion of the graph, an additional bias will be added in the case that the constraints are not fully satisfied, which will encourage the algorithm to explore

the solutions with smaller congestion value which satisfies all the constraints. The final score for state s is then $Q_s = cong(w) + cost(\text{unchecked\_constraints})$.

Furthermore, we assume that the arcs of a graph have a default order, and the metric values corresponding to them are assigned sequentially. Thus, an action a is therefore a choice of metric values for an arc, and the state s is uniquely determined by the metric values already assigned to the arcs. For NRPA, the domain-specific code is uniquely determined by the node of the graph to which the metric is currently to be assigned.

3.3 Improvement

In order to improve the stability of the NRPA algorithm, a stabilized version of NRPA is proposed in [14] to encourage exploration before the adaptation of the policy. During the level 0 of NRPA, instead of running a single playout and use its result as the score, multiple playouts will be performed and only the best result will be used as the score. It improves the average scores for many problems. In our experiments, the stabilized NRPA also achieved better performance than the original NRPA. For brevity, we denote the stabilized NRPA with m playouts as NRPA(m).

We also found that, during the execution of the NRPA, for small and medium-sized graphs, the algorithm tends to prefer exploitation over exploration, which means that the same metric would be obtained many times without exploring new ones. To avoid or limit this behavior, we propose to (i) use a hashtable to record all explored metrics and their scores to avoid recalculation of the congestion and (ii) a *force_exploration* mechanism, which can be of independent interest for the NRPA algorithm. This mechanism works as follows: firstly, all explored metrics w are recorded with their hash codes. Instead of just proposing the metric based on the policy, if one metric has already been explored, a random metric value will be assigned to a random arc of the graph until the generated new metric have never been explored. This simple technique increases the exploration to the maximum, without changing the original NRPA's mechanic. We find that *force_exploration* greatly increases the performance of the NRPA and Stabilized NRPA.

For some graphs, it is difficult to find routing metrics that satisfy all constraints, especially unique path constraints. In such cases, using a unique metric for each arc can greatly increase the proportion of results that satisfy the constraints. However, this limits the number of metrics to be greater than or equal to the number of arcs. This will in many cases increase the proportion of valid solutions, *i.e.* solutions that satisfy the constraints. But as we will show later, in the absence of constraints, this restriction reduces the quality of the solution.

4 Experimental Results

The algorithms are implemented in C++ and the experiments are done on a server (64-core Intel(R) Xeon(R) Gold 5218 CPU), with 125 GB of memory. Only one core is used during the experiments.

4.1 Dataset

The experiments are done on several graphs from SNDlib [29] of different sizes. In addition to these instances, some random graphs are also generated using the same configuration as in [23]. The nodes are generated uniformly in a unit square, and the probability of having an arc between any two nodes is determined by a constant. The capacity of all arcs is set to 1000. We also used Waxman graphs [23] for our test. The probability of having an arc between two nodes is given by:

$$p(u,v) = \alpha e^{\frac{-d(u,v)}{\beta d_{\max}}}$$

where $d(u,v)$ is the Euclidean distance between u and v, $d_{\max}$ is the Maximum Euclidean distance between any two nodes, α and β are parameters which control the density of the graph. The capacities of the arcs are also set to 1000.

For the generated graphs, demands are generated the same way as [23], i.e., the traffic volume D^k for demand k between nodes s^k and t^k is:

$$D^k = \alpha S_{s^k} T_{t^k} C_{(s^k,t^k)} e^{\frac{-d(s^k,t^k)}{2d_{\max}}}$$

where $S_u, T_u \in [0,1]$ are two random numbers for node u, and $C_{(s,t)} \in [0,1]$ is a random number for couple (s,t). Every generated graph is verified to be connected. Table 1 shows the information of the graphs we used in our experiments. Each graph is pre-processed before the computation: all arcs connected to isolated nodes have a pre-determined metric, since their values do not affect the traffic and therefore are not considered again in the Monte Carlo computation.

Table 1. Information of networks: network name, number of nodes, number of arcs, number of demands, total demand, maximum demand

Name	$\|V\|$	$\|A\|$	$\|K\|$	$\sum D^k$	$\max(D^k)$	Name	$\|V\|$	$\|A\|$	$\|K\|$	$\sum D^k$	$\max(D^k)$
abilene	12	30	132	3000002	424969	rand50a	50	132	2450	81419	251
atlanta	15	44	210	136726	7275	rand50b	50	278	2450	86981	249
newyork	16	98	240	1774	42	rand100a	100	278	9900	269535	240
france	25	90	300	99830	1808	rand100b	100	534	9900	307699	228
norway	27	102	702	5348	14	wax50a	50	142	2450	85150	235
nobel-us	14	42	91	5420	324	wax50b	50	298	2450	82208	221
nobel-ger	17	52	121	660	50	wax100a	100	284	9900	331386	270
nobel-eu	28	82	378	1898	54	wax100b	100	492	9900	293799	243
brain	161	332	14311	12.3e9	69.1e6						

4.2 Comparison of the Monte Carlo Algorithms

With both unique path and delay constraints applied, the Monte Carlo search approaches are evaluated during a limited runtime.

During our experiments, we found that executing several different runs was generally better than executing only one long run, because different runs allows

the policy to start over, thus avoiding the algorithm getting stuck at some local minimum. Table 2 shows the average score of each approach on 5 executions. Since for a limited period of time, UCT and NRPA can perform multiple runs of short duration or one long run, we kept both results for comparison. For NRPA and Stabilized NRPA, only force exploration is used.

Table 2. Comparison of the best scores of Monte Carlo Search Algorithms in limited runtime with all constraints. Format: average score of 5 executions (number of executions in which no valid solution was found). "–": no valid solution is found in all executions.

Class	Name	Runtime (min)	$\|W\|$	UCT		NMCS	NRPA		NRPA(10)
				Multiple	Single		Multiple	Single	
SNDlib	abilene	10	50	65.158	87.108	60.905	60.905	60.905	60.905
	atlanta	10	50	2.9608	4.3968	2.3626	2.318	2.318	2.318
	newyork	30	100	0.1255(1)	0.172(4)	0.0978	0.0636	0.062	0.0622
	france	30	100	3.983	–	3.20836	2.892	2.92	2.9268
	norway	60	150	–	–	0.508(2)	0.3054	0.295	0.3042
	nobel-us	10	50	29.48	35.04	25.68	25.2	25.2	25.2
	nobel-ger	10	100	4.44	6.06	4.4	4.4	4.4	4.4
	nobel-eu	30	100	12.08	14.6(3)	10.94	10.7	10.88	10.74
	brain	60	50	1.002	1.0513	0.9848	0.974	0.961	0.98

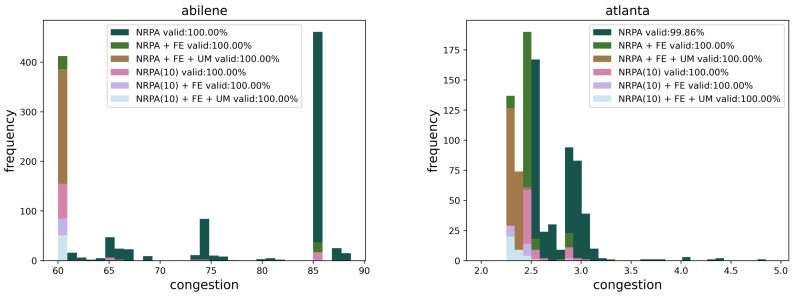
The table very clearly shows that NRPA and its variants outperform the other two MC methods. Moreover, the number of playouts of NRPA does not depend on the size of the graph or the size of the metric space, which makes it easier to scale the algorithm to larger graphs and search spaces. As for NMCS, it performs well with smaller graphs and small search spaces, but does not scale well to larger cases. Therefore, in the subsequent experiments, we will only consider the use of NRPA-based algorithms.

Since there are many variants for NRPA, we first investigated the effect of these techniques. Figure 2 shows the distribution of the scores on several SNDlib graphs with different techniques of NRPA. The more the distribution is concentrated around the low values, the higher the chance of getting lower congestion values, thus the better this configuration is.

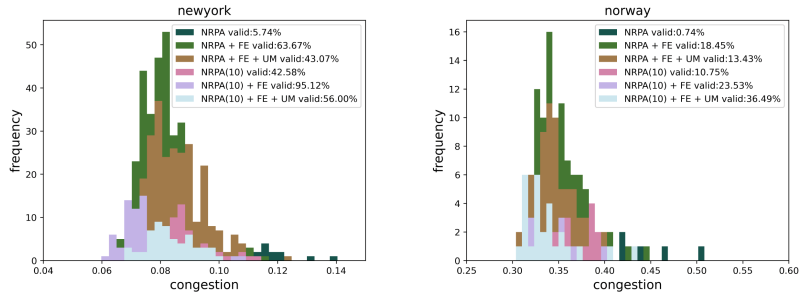
The distribution clearly demonstrates the improvement of *force_exploration* for NRPA and stabilized NRPA, while stabilized NRPA greatly increases the ratio of valid solutions. However the effect of unique metrics is not as obvious, and this extra constraint can sometimes make it more difficult to find better results.

4.3 Impact of the Metric Space

Although in many previous studies of heuristics, the metric space is generally a continuous set of integers, in the course of our research, we found that the



(a) $|W|$=50, t=10 min (b) $|W|$=50, t=10 min



(c) $|W|$=100, t=30 min (d) $|W|$=150, t=60 min

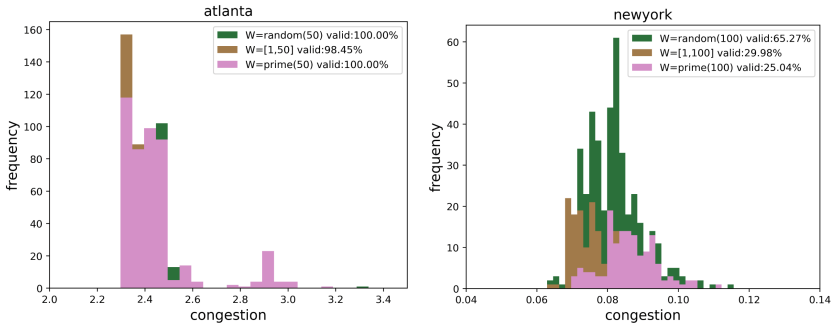
Fig. 2. Distribution of the congestion values with all constraints on SNDlib graphs

metric space has an important impact on the performance of the algorithm. For example, compared to a continuous set of integers or the set of prime numbers, a set of random numbers of the same size usually gives a much higher rate of valid solutions. Figure 3 shows an example of the distribution of scores obtained with different metric spaces of the same size. The random numbers are uniformly pre-generated in $[1, 65535]$, and remain the same during the experiment.

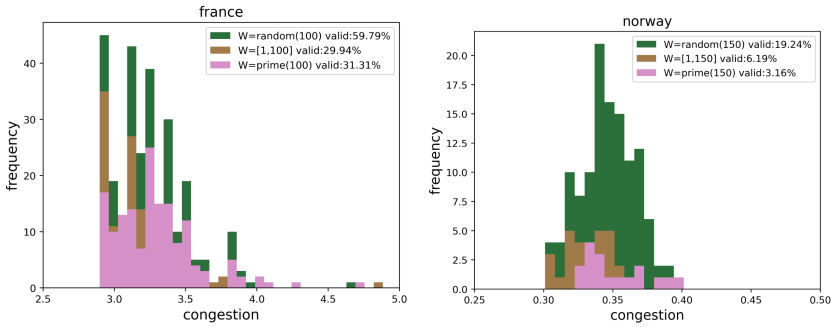
We also find out that for most graphs, when no constraints are applied, a smaller metric space helps the algorithm to converge better because the algorithm can better explore the search space. Figure 4 shows the convergence of different metric space sizes. However when unique path and delay constraints are applied, a small metric space can make it more difficult to find a metric that induces a valid solution. Therefore, increasing the search space can greatly increase the chance of obtaining a solution that satisfies all constraints. Figure 5 shows the influence of the metric space on the percentage of valid solutions.

4.4 Comparison

In this test, we compare the congestion value computed by our algorithm after a fixed amount of time with the congestion values obtained via other common approaches. The first approach for the problem, which is also the most basic approach is the UnitOSPF, which assigns all arcs the same unit metric value.



(a) $|W|$=50, t=10 min (b) $|W|$=100, t=30 min



(c) $|W|$=100, t=30 min (d) $|W|$=150, t=60 min

Fig. 3. Distribution of the congestion values with different metric spaces of the same size using NRPA with *force_exploration*

InvCapOSPF is another approach recommended by Cisco, which sets the metric inversely proportional to the arc's capacity. However, in many graphs the capacity on all arcs is the same, so in many cases this method will give the same results as UnitOSPF. We also compare our algorithm with the local search IGP-WO implemented in [27] and based on [23]. We slightly modify the objective function of IGP-WO to minimize the congestion and, when considering the unicity constraint, add a high penality for solutions that violate that constraint. We did not further modify the algorithm to take into account the delay constraint as this would require deeper modifications and understanding of the implementation of IGP-WO.

In order to evaluate the quality of the heuristic solutions, we compare them with the optimal value obtained using the compact formulation of the Maximum Concurrent Flow (MCF) problem. It is not hard to see that any optimal solution for MCF is a lower bound for Min-Con-SPR (denoted LPLB for Linear Programming Lower Bound). Indeed, a solution of a MCF instance defines routing paths for each demand that are not constrained to follow the ECMP rule or being induced by shortest paths w.r.t some metric.

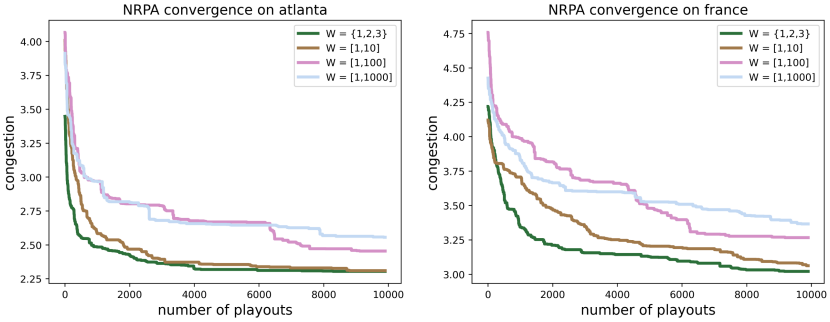


Fig. 4. Convergence of NRPA without constraints. The score is averaged on 10 runs.

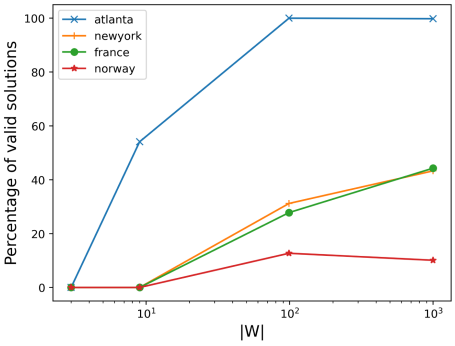


Fig. 5. Size of the metric space has a positive impact on the valid solution ratio when constraints are applied. The result is averaged on 1000 runs

Table 3 shows the comparison of the results. To keep the results consistent across configurations, for NRPA, the metric space size and computation time of each graph are consistent with those shown in Table 2 for the constrained cases. For all generated graphs, the metric space size is 250, and the computation time is 30 min for graphs of 50 nodes, 60 min for graphs of 100 nodes. However when no constraints are applied, the metric space for all graphs is set to [1,3] for better performance. The scores are averaged on 5 executions.

For random graphs and waxman graphs, the applied delay constraint is often too restrictive so that no valid solutions can be found. In this paper, we propose an automatic delay relaxation mechanism that allows the NRPA algorithm to relax the delay constraint when it is difficult to find a valid solution.

In the initial state, the delay constraint is actually defined as the length of the shortest path (in terms of arcs) plus one for each demand. We relax the constraint to shortest path plus c, where $c \in N^+$. Assuming that the computation has a target time or total number of runs T, we divide all the computations into different phases. At each stage, we count the total number of runned playouts, the number of valid solutions and the number of solutions that violate the delay constraint. After $T \times (1 - \frac{1}{2^c})$ time or iterations, if the percentage of valid solutions

is less than 5% and the percentage of results that violate the delay constraint is greater than 10%, we move to the next stage, where all statistics are recalculated and $c = c + 1$, until c reaches a maximum provided value c_{max}.

Table 3. Maximum congestion value of state-of-the-art heuristics and our NRPA. For each constraint configuration ("Without constraints", "Unicity" and "All" *i.e.* both "Unicity" and "Delay") and each heuristic (if available for the given configuration), we show the congestion induced by the computed weights. This value is in bold if it is the best one among those returned by the other heuristics (w.r.t the configuration). In addition, a value is followed by * if equal to the lower bound LPLB (which is reported in the last column). Finally, an entry $c = i$ indicates the minimum value i of c for which we were able to find a solution when considering all constraints.

Name	Without constraints				Unicity		All	LPLB
	Unit OSPF	InvCap OSPF	IGP-WO [27]	NRPA	IGP-WO	NRPA	NRPA	
abilene	187.55	89.48	60.42	**60.412**	**60.41**	**60.41**	60.90	60.411
atlanta	3.26	3.37	**2.22**	**2.22**	**2.29**	**2.29**	2.32	2.18
newyork	0.076	0.076	**0.051**	0.053	**0.062**	0.065	0.064	0.045
france	4.12	4.12	**2.53**	2.56	**2.88**	**2.88**	2.89	2.41
norway	0.42	0.42	**0.28**	0.29	**0.29**	0.30	0.31	0.27
nobel-us	37.15	37.15	**24.4**	24.7	**24.7**	**24.7**	25.2	24.2
nobel-ger	5.54	5.54	3.9	**3.89**	**4.4**	**4.4**	4.4	3.87
nobel-eu	13.31	13.31	10.68	**10.67***	**10.7**	**10.7**	10.7	10.67
brain	1.415	1.415	0.962	**0.903***	**0.972**	**0.972**	0.974	0.903
rand50a	7.9	7.9	**5.55**	5.77	**5.84**	5.92	5.96(c = 2)	5.55
rand50b	**2.88***	**2.88***	**2.88***	**2.88***	–	**2.88**	2.88(c = 3)	2.88
rand100a	15.71	15.71	10.42	**9.59**	–	**10.35**	10.76(c = 4)	9.35
rand100b	4.15	4.15	4.38	**3.85**	–	**6.06**	5.94(c = 5)	3.76
wax50a	6.46	6.46	**4.63**	4.66	**4.665**	4.67	4.71(c = 2)	4.59
wax50b	**2.279***	**2.279***	2.284	**2.279***	–	**2.279**	2.279(c = 3)	2.279
wax100a	17.46	17.46	15.049	**15.048**	–	**15.049**	15.049(c = 4)	15.048
wax100b	5.51	5.51	4.14	**4.04**	–	**5.86**	5.91(c = 5)	3.44

The results show that NRPA performs very well for all three different constraint configurations on all sizes of graphs and is very close to the lower bound. Compared to local search, our algorithm gives better results in most cases. At the same time, only very little computational time and resources are used. The different runs can be computed in parallel, which substantially improves the running time of NRPA.

The proposed automatic delay relaxation mechanism is not the best way to solve the problem of minimizing the congestion of the graph while keeping the delay minimized. Nevertheless, we obtained rather encouraging results that show the strong adaptability of our approach and a promising potential for solving even more difficult variants of the problem (single-link failure, oblivious routing, capacity planning, ...).

4.5 Random Dense Graphs

For dense networks, the number of potential routing paths increases rapidly, which makes the problem even harder to solve in particular with the unicity constraint, as observed in [5]. We show that NRPA can easily scale up to graphs with large amount of nodes and arcs. So we generated ten random graphs of different sizes with the same generation mechanism as described in Sect. 4.1, and the traffic between two nodes are generated with a probability of 0.1. The largest graph contains 1000 nodes and 99450 arcs.

Figure 6 shows the scores of the random graphs without constraints. The metric space is set to [1,3] for both local search and NRPA algorithms, and the scores of NRPA are averaged on 10 executions.

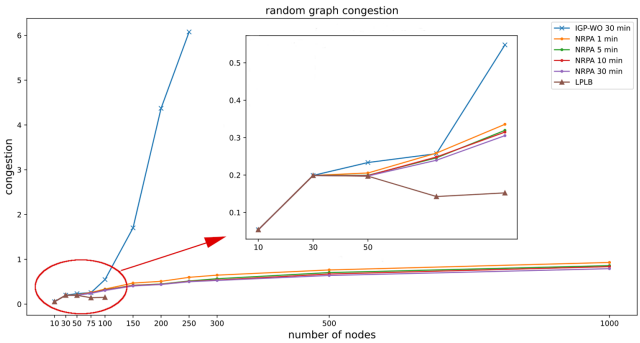


Fig. 6. Congestion with respect to the number of the nodes

Regarding the LPLB bound, with a compact formulation, we were not able to compute the lower bounds for instances larger than 100 nodes because the random graphs are more dense and thus the problem is too large to be solved. In future experiments, using a non-compact path formulation of the MCF may greatly improve the chances of obtaining lower bounds for larger graphs.

The results show that on large graphs, local search does not provide acceptable results within an execution time of 30 min. On the contrary, even on graphs with thousands of nodes, our method still gives reasonable results in a very short time.

5 Conclusion

In this work we applied for the first time the Monte Carlo Search approach in the context of setting efficient weights in IP networks, in particular for the Min-Con-SPR problem. The principle of the Monte Carlo Search algorithm is to learn a policy online on each instance using nested levels of best solutions. We compare several Monte Carlo methods and propose the most appropriate to the target problem. Experiments show that for instances from the literature

our approach is comparable with the existing ones. Nevertheless, for graphs of larger size, our approach outperforms the local search heuristics and gives results close to the lower bound. At the same time, this approach can be easily extended for problems with additional constraints and is not sensitive to the size of the graph or the size of the search space in particular the number of available weights, giving it a large range of applications. Furthermore, for the unsplittable case, this method may provide optimal solutions (or close to the optimal) for instances where exact approaches fail, especially for dense graphs [5]. For some instances where it is not possible to find a way to satisfy all the constraints, we also propose a mechanism for automatically relaxing the constraints. Another algorithm specifically aimed at optimizing congestion with the lowest possible delay constraint will be the direction of subsequent research.

References

1. Altin, A., Fortz, B., Thorup, M., Ümit, H.: Intra-domain traffic engineering with shortest path routing protocols. Ann. Oper. Res. **204**(1), 65–95 (2013). https://doi.org/10.1007/s10479-012-1270-7
2. Altin, A., Fortz, B., Ümit, H.: Oblivious OSPF routing with weight optimization under polyhedral demand uncertainty. Networks **60**(2), 132–139 (2012)
3. Benhamiche, A., Chopin, M.: Toward scalable algorithms for the unsplittable shortest path routing problem. Research report, Orange Labs (2020)
4. Bley, A.: Approximability of unsplittable shortest path routing problems. Networks **54**(1), 23–46 (2009)
5. Bley, A.: An integer programming algorithm for routing optimization in IP networks. Algorithmica **60**(1), 21–45 (2011)
6. Bley, A., Fortz, B., Gourdin, É., Holmberg, K., Klopfenstein, O., Pióro, M., Tomaszewski, A., Ümit, H.: Optimization of OSPF routing in IP networks. In: Koster, A., Muñoz, X. (eds.) Graphs and Algorithms in Communication Networks: Studies in Broadband, Optical, Wireless and Ad Hoc Networks. An EATCS Series, pp. 199–240. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-02250-0_8
7. Bouzy, B.: Monte-Carlo fork search for cooperative path-finding. In: Cazenave, T., Winands, M.H.M., Iida, H. (eds.) CGW 2013. CCIS, vol. 408, pp. 1–15. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-05428-5_1
8. Buriol, L.S., Resende, M.G.C., Ribeiro, C.C., Thorup, M.: A hybrid genetic algorithm for the weight setting problem in OSPF/IS-IS routing. Networks **46**(1), 36–56 (2005)
9. Cazenave, T.: Nested Monte-Carlo search. In: Boutilier, C. (ed.) IJCAI, pp. 456–461 (2009)
10. Cazenave, T., Fournier, T.: Monte Carlo inverse folding. In: Monte Search at IJCAI (2020)
11. Cazenave, T., Lucas, J., Triboulet, T., Kim, H.: Policy adaptation for vehicle routing. Ai Commun. (2021)
12. Cazenave, T., Negrevergne, B., Sikora, F.: Monte Carlo graph coloring. In: Monte Search at IJCAI (2020)
13. Cazenave, T., Saffidine, A., Schofield, M.J., Thielscher, M.: Nested Monte Carlo search for two-player games. In: AAAI, pp. 687–693 (2016)

14. Cazenave, T., Sevestre, J.B., Toulemont, M.: Stabilized nested rollout policy adaptation. In: Monte Search at IJCAI (2020)
15. Cazenave, T., Teytaud, F.: Application of the nested rollout policy adaptation algorithm to the traveling salesman problem with time windows. In: Hamadi, Y., Schoenauer, M. (eds.) LION 2012. LNCS, pp. 42–54. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-34413-8_4
16. Edelkamp, S., Gath, M., Cazenave, T., Teytaud, F.: Algorithm and knowledge engineering for the TSPTW problem. In: 2013 IEEE Symposium on Computational Intelligence in Scheduling (SCIS), pp. 44–51. IEEE (2013)
17. Edelkamp, S., Gath, M., Greulich, C., Humann, M., Herzog, O., Lawo, M.: Monte-Carlo tree search for logistics. In: Clausen, U., Friedrich, H., Thaller, C., Geiger, C. (eds.) Commercial Transport. LNL, pp. 427–440. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-21266-1_28
18. Edelkamp, S., Gath, M., Rohde, M.: Monte-Carlo tree search for 3D packing with object orientation. In: Lutz, C., Thielscher, M. (eds.) KI 2014. LNCS (LNAI), vol. 8736, pp. 285–296. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-11206-0_28
19. Edelkamp, S., Greulich, C.: Solving physical traveling salesman problems with policy adaptation. In: 2014 IEEE Conference on Computational Intelligence and Games (CIG), pp. 1–8. IEEE (2014)
20. Edelkamp, S., Tang, Z.: Monte-Carlo tree search for the multiple sequence alignment problem. In: SOCS 2015, pp. 9–17. AAAI Press (2015)
21. Ericsson, M., Resende, M.G.C., Pardalos, P.M.: A genetic algorithm for the weight setting problem in OSPF routing. J. Comb. Optim. **6**(3), 299–333 (2002). https://doi.org/10.1023/A:1014852026591
22. Fortz, B.: Applications of meta-heuristics to traffic engineering in IP networks. Int. Trans. Oper. Res. **18**(2), 131–147 (2011)
23. Fortz, B., Thorup, M.: Increasing internet capacity using local search. Comput. Optim. Appl. **29**, 13–48 (2000). https://doi.org/10.1023/B:COAP.0000039487.35027.02
24. Fortz, B., Thorup, M.: Robust optimization of OSPF/IS-IS weights. In: INOC, pp. 225–230 (2003)
25. Kinny, D.: A new approach to the snake-in-the-box problem. In: ECAI 2012, pp. 462–467. IOS Press (2012)
26. Kocsis, L., Szepesvári, C.: Bandit based Monte-Carlo planning. In: Fürnkranz, J., Scheffer, T., Spiliopoulou, M. (eds.) ECML 2006. LNCS (LNAI), vol. 4212, pp. 282–293. Springer, Heidelberg (2006). https://doi.org/10.1007/11871842_29
27. Leduc, G., et al.: An open source traffic engineering toolbox. Comput. Commun. **29**(5), 593–610 (2006)
28. Méhat, J., Cazenave, T.: Combining UCT and Nested Monte Carlo search for single-player general game playing. IEEE TCIAIG **2**(4), 271–277 (2010)
29. Orlowski, S., Pióro, M., Tomaszewski, A., Wessäly, R.: SNDlib 1.0-survivable network design library. Networks **55**(3), 276–286 (2010)
30. Poulding, S.M., Feldt, R.: Heuristic model checking using a Monte-Carlo tree search algorithm. In: GECCO, pp. 1359–1366 (2015)
31. Rimmel, A., Teytaud, F., Cazenave, T.: Optimization of the Nested Monte-Carlo algorithm on the traveling salesman problem with time windows. In: Di Chio, C., et al. (eds.) EvoApplications 2011. LNCS, vol. 6625, pp. 501–510. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-20520-0_51
32. Rosin, C.D.: Nested rollout policy adaptation for Monte Carlo Tree search. In: IJCAI, pp. 649–654 (2011)