Budget Limited Spatial Network Improvement Using Monte Carlo Search

Milo Roucairol[a] **and Tristan Cazenave**[a;*]

[a]LAMSADE, Université Paris Dauphine - PSL
ORCiD ID: Milo Roucairol https://orcid.org/0000-0002-7794-5614,
Tristan Cazenave https://orcid.org/0000-0003-4669-9374

Abstract. We propose a comparison of a large selection of state-of-the-art deterministic and Monte Carlo Search (MCS) algorithms on the budget-limited network optimization problem. Including a new one producing better results, and different approaches to simplifying this problem. We show that not one algorithm can be the only answer for optimizing both efficiency and robustness, but that a simple heuristic can perform well for the optimization of robustness, and that the LNMCS with greedy playout is a reliable choice for optimizing efficiency on 100 nodes large graphs. LNMCS and other algorithms are able to outperform the previous state of the art in all situations and variations of the problem encountered. We provide results for procedurally generated graphs and real-world ones, and show that results obtained on synthetic graphs match with real-world graphs.

1 Introduction

Optimizing transportation networks is one of the most important real-world problems engineers have to face. Critical infrastructure, such as the Internet or roads, efficiencies and robustness rely on optimizing these types of networks.

Search algorithms are already making most of the State of the Art on this problem. More precisely, stochastic search algorithms dominate the network optimization problem due to the large search space. We identified two families of stochastic search algorithms for network optimization: genetic algorithms [9], [15], and Monte Carlo Search algorithms [2].

Whether it be communications or transportation, the optimization must respect constraints such as budget or topology. In this paper, we compare the performances of multiple algorithms over synthetic and real world instances of the problem. Classic deterministic ones such as Beam search or Best First Search (BFS). Against Monte Carlo Search ones, namely Upper Confidence bounds applied to Trees (UCT) and others such as Nested Monte Carlo Search (NMCS), Nested Rollout Adaptation (NRPA), and Rapid Action Value Estimation (RAVE). As transportation and communication networks have to be efficient and resilient, we chose to optimize them over one metric focused on robustness, and another focused on efficiency.

* Corresponding Author. Email: tristan.cazenave@lamsade.dauphine.fr .

2 The Budget Limited Spatial Network Improvement Problem

Our instance of the Network Design Problem (NDP) [18] consists of a pre-existing weighted graph with infinite capacity. The weight on the edges represents the cost of going to the node on one side of the edge from the node on the other side of the edge. Here, the cost is the distance between the two nodes. It is the kind of graph on which we can use the Dijkstra and Floyd-Warshall algorithms. We do not look at the flow optimization in this paper, only the topological properties.

Our goal is, given such a graph, to optimize selected metrics on it. The optimization process starts from a graph and adds new edges until the budget is not exhausted, each edge added subtracts its length to the budget. The actions a search algorithm can select are the edges that are absent from the graph. The final states are states where the budget is too low to add any new edge. Here we chose to maximize specific definitions of the efficiency [8] and of the robustness [4].

Efficiency is defined as in Latora's work [8] : $E(G) = \frac{1}{N(N-1)} \sum_{i \neq j} \frac{1}{sp(i,j)}$ with N the size of the graph, and $sp(i, j)$ the shortest path between vertices i and j. Similarly to [2], we divide this value by the ideal efficiency, the efficiency of the complete graph, to obtain a value between 0 and 1.

However, we decided to use another metric for robustness than the one used in [2]. We think this metric for robustness is too computationally costly (we can not compare our results directly with theirs anyway, see section 4). We instead decided to use the much less costly spectral radius of the adjacency matrix λ_1. It is the largest eigenvalue and is described as a powerful robustness estimation in [4], in the words of this survey:

"The spectral radius is closely related to the path capacity or loop capacity of the graph. That is, the number of walks of length k (k = 2, 3, 4...) gives an indication of how well connected the graph is. If the graph has many loops and paths, then the graph is well connected i.e., larger λ_1."

"As a robustness measure, a larger λ_1 indicates a more robust graph to random failures and attack, along with increased susceptibility to virus propagation."

This is not the only relevant robustness metric present in [4]. It is also stated that the average distance between stops (i.e. the efficiency) can be a good robustness metric, and that spectral-based techniques are scalable to larger graphs, which is one of our goals here.

2.1 Synthetic instances

To compare our algorithm, we need instances of various sizes. We used the same method as used in [2], which itself was from [7]. This method is as follows:
(1) place a node u with random positions in $[0, 1]^2$
(2) if there are other nodes, connect this node to each of them (v) given the probability $p(u, v) = \beta e^{-\alpha d(u,v)}$
(3) if u was not connected remove it
(4) if there's less than the desired amount of nodes, go to (1)

We use the same parameters as in [2]: $\alpha = 10$ and $\beta = 0.001$

The networks generated this way have the advantage of resembling real-world networks, as you can see in figure 1. However, they feature overlapping edges that are absent from real-world networks.

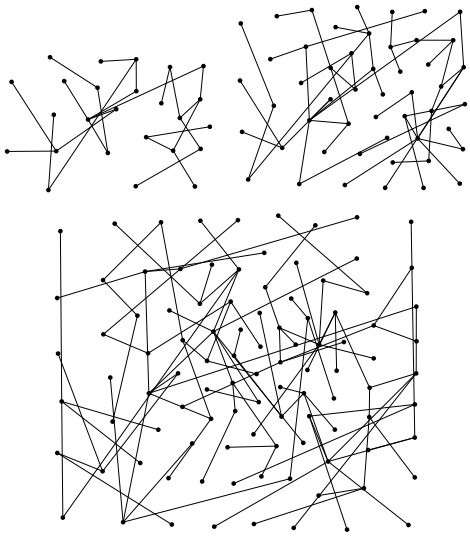

Figure 1: Graphs of size 25, 50, 100 generated by seed 0

3 Algorithms

All algorithms presented in the following subsections were selected for being the state of the art in tree search, Monte Carlo and deterministic. Although many other state-of-the-art deterministic algorithms exist.

3.1 Playout

Playouts or Rollouts are the core elements of most Monte Carlo Search (MCS) algorithms. They are used to evaluate a state by simulating the rest of the generation from it.

Playouts are usually uniformly random and return a terminal state. However, if the problem admits any state encountered during the simulation as a potential solution, the best state encountered can be returned. The playout can also not be uniformly random, but can be guided by the immediate gain. Finally, the playouts can be guided by a learned policy like with the Nested Rollout Policy Adaptation (NRPA) algorithm (a rollout is another name for playout).

3.2 BEAM

BEAM search is a simple baseline tree search algorithm. It only takes one integer parameter, the width w. It necessitates having access to a reliable evaluation of the search space states; otherwise, the evaluation can be made with playouts. The algorithm keeps in memory w nodes, opens and evaluates all the children of these nodes, and then replaces the previous w nodes with the w best nodes recently opened. It has the advantage of forcing progress deep into the tree.

A beam search with $width = 1$ is what we call a greedy playout. A beam search of width w is usually w times more computationally heavy than a beam search of width 1; thus, greedy playouts can be replaced by beam searches in many algorithms with a linear increase in computation costs.

3.3 GBFS

Greedy Best First Search is another simple yet effective baseline tree search algorithm [3]. It necessitates having access to a reliable way to evaluate nonterminal states, using playouts as a substitute if it is not possible, like Beam search. The algorithm selects the node with the best evaluation from a queue, evaluates all its children, and inserts them into the queue according to their evaluations. Thus, GBFS is guaranteed to explore only once all the nodes in the search tree, unlike BEAM search, which can miss states with great valuations. Missing states with great valuations is usually unavoidable and not a shortcoming since the problems we are facing are generally NP-Hard and could never be solved exhaustively anyway.

3.4 UCT

The Upper Confidence Bound applied to Trees is the most widely used instance of Monte Carlo Tree Search, and the default algorithm to which one goes when using Monte Carlo Methods. It, or variants of it, is used in groundbreaking applications such as Deepmind's Alphago [16] or AstraZeneca's Aizynthfinder [6], and other [17].

Contrary to BEAM and GBFS, MCTS does not need a way to evaluate any state from the search tree, and only needs to evaluate final states. This is especially useful in games, like chess, where the function to evaluate a game in progress is not trivial. But even when such a function is available, MCTS algorithms are generally better performing because they are more capable of avoiding local maximum and other traps that come with a noisy search space.

UCT and all the other MCTS algorithms share the same 4 phases:
(1) selection: until an unknown node is opened, go down the search tree according to the exploration/exploitation formula. (2) expansion: add a new node to the search tree (3) simulation: evaluate the newly added node, using playouts to get a terminal state (4) backpropagation: use the simulation result to update the values of all the search tree nodes visited during the selection phase

In UCT's case, the exploration/exploitation formula for each move m from a state s is: $\frac{w}{n} + c * \sqrt{\frac{lnN}{n}}$

Where w is the sum of all scores obtained after playing m from s, n is the number of times m was played from s, and N is the number of times s was visited during the selection phase. c is a constant (usually $c \approx 1$).

3.5 RAVE and GRAVE

Rapid Action Value Estimation (RAVE) is an MCTS algorithm derived from UCT and introduced by Sylvain Gelly and David Silver [5]. The main difference with UCT is that RAVE generalizes the value of moves over the entire search tree (for example, if a move generally leads to better results, then it may be favored even if in the

UCT-like subtree it led to worse results). This is adapted to problems where the order of the moves is less important, for example, in go and not in chess. We think the network optimization problem is appropriate.

The Generalized Rapid Action Value Estimation (GRAVE) is a generalization of RAVE [1]. Unlike RAVE, when deciding which move to play, it inherits a policy from the last parent move whose children experienced more than ref playouts for more localized generalization. It can also be used in conjunction with a move selection heuristic.

3.6 NMCS

Nested Monte Carlo Search is another type of Monte Carlo Search algorithm, different from MCTS like UCT, PUCT, RAVE, and GRAVE in their iterative natures; NMCS is recursive.

The NMCS calls lower-level NMCS on all of the currently available moves from the current state. Each NMCS returns the best path it found to optimize the value of the state. The higher-level NMCS then executes the first action from the best path it has in memory and calls new lower-level NMCS on the available moves from the resulting state.

Compared to UCT, NMCS has the advantage of optimizing at any depth of the search tree and not only near the root. It generally shows better results on optimization problems [12] [13].

3.7 LNMCS

The Lazy Nested Monte Carlo Search presented in [12] is a variant of NMCS made to address one of its shortcomings. An NMCS of level l requires computing as many NMCS of level $l-1$ as the number of moves available from the state, and then repeating that for each level of depth of the search tree. The computation time of the NMCS increases greatly with the level, an NMCS of level over 3 or even 2 can be too computationally costly depending on the problem.

Under the assumption that some moves doom the lower-level NMCS to underwhelming results, we decide to reintroduce the exploration-exploitation dilemma present in MCTS to the NMCS, and prune some of the lower-level NMCS based on cheap and relative evaluations using playouts.

Before calling a lower-level LNMCS, the available moves are each sampled with b playouts. If the mean of the evaluations of a move is inferior to the mean of all the evaluations made at that depth tr plus the rate r times the difference between the best evaluation ever encountered on that depth $trmax$ and the mean of all the evaluations made on that depth tr, then it is pruned: a single playout is launched instead of a lower-level LNMCS. For example in Figure 2, the middle and right moves are sampled with good enough results to pass the threshold and their lower-level LNMCS are called, while the leftmost move has poor sampling and is pruned.

This means that a rate r of 0 prunes all the moves inferior to the mean of the evaluations at a certain depth.

The pseudocode for LNMCS is available in [12].

3.8 NRPA

The Nested Rollout Policy Adaptation is a MCTS algorithm introduced by Christopher Rosin [11] in 2011. It is derived, but very different, from the NMCS. It uses the nesting not to progress deeper into the tree, but to contribute hierarchically to a policy that is learned from playouts (or rollouts) to guide future playouts.

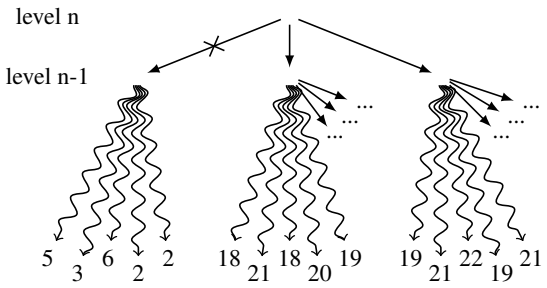

Figure 2: Level n LNMCS pruning a search subtree and launching n-1 LNMCS on surviving search subtrees.

4 Results

4.1 Preliminary results

4.1.1 Variation among the synthetic instances

Before diving into the performances of our algorithms on the synthetic benchmark instances, we think it is crucial to sample these possible synthetic instances to know more about their depth and potential scores. To do so, we generated 20 synthetic instances of size 25 with $\alpha = 10$ and $\beta = 0.001$ and maximized their efficiency using an exhaustive algorithm with a budget of 0.1 times the total cost of the starting edges as in [2].

With as little as 15206 and as much as 18222026 explored states, even small 25 nodes graphs can take hours to be explored exhaustively, or a few minutes. The search space size can vary greatly with a standard deviation of 4183678.3, mainly due to extreme outliers.

The starting efficiency and their improvement show a significant standard deviation too.

With such variations among all the parameters surrounding the synthetic instances, it appears necessary to compare our algorithms on multiple instances and share them to help with reproducibility and improve the relevance of said results. There seems to be no correlation between the size of the search tree and the efficiency gain over this small sample, however, the start efficiency and the efficiency gain seem to sum around 0.8 with these parameters.

Given [2] results show almost no variation, we assume they realized all their experiments on a graph generated from a unique seed.

4.1.2 Hyperparameter tuning and algorithmic choice

Before diving into lengthy experiments, we first need to quickly evaluate the algorithms likely to perform well and how much the results would vary

We launched UCT with $c = 1.0$ and NMCS with $level = 2, 3$, 10 times over the graph of size 50 generated by the seed 0 with the previously mentioned parameters and budget with a timeout of 600 seconds, the initial score is 0.43899. During our 10 experiments, we obtain the final efficiencies featured in tables 2. The experiments in this subsection were performed on an Intel i5-6600K 3.50GHz CPU, which is different from the CPU used in Section 4.2.

It is interesting to note that UCT peaks around the 120th second, and NMCS continues to find better results after the 400th second.

We do not include the standard deviation in subsequent results as it is always between 0.005 and 0.015 and impairs readability. We think it is low enough to justify using the means over 10 runs.

seed	search tree size	start efficiency	efficiency gain
0	180557	0.575	0.222
1	453598	0.498	0.304
2	1118035	0.511	0.279
3	19353	0.600	0.190
4	3990749	0.423	0.354
5	16593	0.585	0.203
6	196488	0.289	0.523
7	451586	0.427	0.301
8	6662608	0.431	0.349
9	15206	0.158	0.571
10	849969	0.604	0.226
11	18222026	0.305	0.498
12	202226	0.468	0.295
13	744556	0.475	0.305
14	144717	0.491	0.312
15	1538641	0.432	0.322
16	247973	0.527	0.273
17	3326338	0.540	0.226
18	443276	0.455	0.340
19	334322	0.529	0.331
Mean	$1.957e6$	0.466	0.321
Std dev	$4.183e6$	0.112	0.103

Table 1: Differences in search tree sizes, starting and best values among 20 synthetic graphs of size 25 with $\alpha = 10$ and $\beta = 0.001$

experiment	UCT	NMCS 2	NMCS 3
1	0.56436	0.57813	0.59877
2	0.55633	0.58891	0.59768
3	0.56271	0.59257	0.58098
4	0.55491	0.57611	0.58943
5	0.55557	0.58085	0.57400
6	0.56734	0.58652	0.58554
7	0.55697	0.57866	0.57649
8	0.57557	0.60954	0.58618
9	0.55668	0.60097	0.59544
10	0.56689	0.60194	0.57162
Mean	0.56173	0.58942	0.58561
Std dev	0.00682	0.01158	0.00980

Table 2: Preliminary comparison between NMCS and UCT and evaluation of the variance on one seed

4.2 On the synthetic instances

4.2.1 Experimental setup

To evaluate each algorithm, we launch each of them 10 times over 3 different sizes: 25, 50, and 100. We then repeat this experiment 5 times, for each graph generating seed from 0 to 4. In tables 3, 4, and 5 we display the results for directly solving the efficiency problem with each algorithm and each size.

In tables 6, 8, and 7 we try new approaches to try to maximize the final efficiency.

In tables 12, 14, and 13 we apply the previously best methods on the robustness problem and compare it against few representative baseline methods.

As seen in Figure 1, these networks seem to imperfectly model networks like streets, since the edges can overlap.

These experiments were made with Rust 1.59, on an Intel Core i7-11850H 2.50GHz using a single core.

We compare the following algorithms:

- UCT, with $c = 1$ as the baseline MCTS algorithm
- GBFS, a baseline greedy deterministic algorithm
- BEAM, with widths of 10, 50, and 100, another baseline greedy deterministic algorithm
- NMCS, with $l = 2$ and another with $l = 3$ given its good performances and relative simplicity
- LNMCS, with the default hyperparameters $l = 3$, $r = 0.8$ and $p = 3$ as it usually improves over NMCS [14]
- NRPA, with $l = 3$
- RAVE, as a recent improvement over UCT
- GRAVE, $ref = 50$ is shown to be a good value in [1]
- GRAVE B with $ref = 50$, $bias = 10$ and using the cost-effectiveness ($value_{change}/cost$) as a move selection heuristic.

4.2.2 Solving the problem directly

seed	0	1	2	3	4
start value	0.575	0.498	0.511	0.600	0.423
BFS	**0.797**	**0.802**	**0.791**	**0.790**	**0.778**
BEAM 10	0.768	0.757	0.736	0.778	0.749
BEAM 50	0.781	**0.802**	0.747	**0.790**	0.770
BEAM 100	0.766	0.769	0.786	**0.790**	0.770
LNMCS	0.755	0.754	0.729	0.764	0.763
NMCS 2	0.769	0.752	0.757	0.772	0.752
NMCS 3	0.791	0.786	0.781	0.789	0.764
NRPA	0.792	0.776	0.770	**0.790**	0.719
UCT	0.724	0.748	0.677	0.746	0.770
RAVE	0.711	0.612	0.687	0.716	0.623
GRAVE	0.704	0.624	0.680	0.702	0.641
GRAVE B	0.708	0.632	0.693	0.709	0.611

Table 3: Final efficiencies found for each algorithm on each graph of size 25 after 600s

seed	0	1	2	3	4
start value	0.438	0.398	0.439	0.445	0.373
BFS	0.607	0.619	0.623	0.639	0.587
BEAM 10	0.614	0.543	0.583	0.668	0.587
BEAM 50	**0.659**	0.624	**0.659**	0.687	0.604
BEAM 100	0.655	0.620	0.642	**0.700**	**0.627**
LNMCS	0.647	**0.632**	0.637	0.681	0.621
NMCS 2	0.628	0.583	0.637	0.673	0.561
NMCS 3	0.641	0.589	0.646	0.692	0.588
NRPA	0.644	0.577	0.650	0.677	0.570
UCT	0.590	0.542	0.602	0.580	0.541
RAVE	0.545	0.510	0.547	0.592	0.487
GRAVE	0.556	0.505	0.570	0.589	0.518
GRAVE B	0.558	0.506	0.557	0.589	0.513

Table 4: Final efficiencies found for each algorithm on each graph of size 50 after 600s

As you can see in Table 1, a network size of 25 is small enough for the BFS to find the optimal value with all seeds except seed 2, which features a large search tree. The optimal value is found on seed 4 despite an even larger search tree. Beam search features good results but they are not optimal, the problem cannot always be solved optimally by greedy playouts. Among the Monte Carlo algorithms, NMCS 3 and NRPA show the best results, the UCT family is lagging behind the nested family. Over 10 runs, the NRPA managed to always find the optimal solution on seed 3, which is impressive for a Monte Carlo algorithm.

With a network size of 50, the BFS is no longer able to explore most of the tree, and larger widths are required for the beam search to produce good results. LNMCS becomes the best algorithm among the MCTS (except on seed 2). When the search tree size increases, pruning bad subtrees becomes more efficient.

As the size of the search tree increases further with networks of size 100, in Table 5, LNMCS becomes the best algorithm and is only outperformed by BFS on seed 3 and BEAM 10 on seed 2. BEAM 10 being a very close second is interesting because it is both a simple

seed	0	1	2	3	4
start value	0.307	0.343	0.357	0.382	0.329
BFS	0.456	0.464	0.477	**0.511**	0.444
BEAM 10	**0.459**	0.476	**0.517**	0.484	0.429
BEAM 50	0.440	0.436	0.449	0.467	0.411
BEAM 100	0.416	0.421	0.439	0.458	0.392
LNMCS	**0.459**	**0.480**	0.497	0.505	**0.461**
NMCS 2	0.443	0.451	0.476	0.479	0.425
NMCS 3	0.437	0.453	0.473	0.479	0.431
NRPA	0.446	0.451	0.471	0.484	0.426
UCT	0.406	0.424	0.441	0.449	0.399
RAVE	0.405	0.412	0.441	0.449	0.386
GRAVE	0.406	0.420	0.437	0.469	0.409
GRAVE B	0.373	0.425	0.422	0.456	0.398

Table 5: Final efficiencies found for each algorithm on each graph of size 100 after 600s

seed	0	1	2	3	4
start value	0.307	0.343	0.357	0.382	0.329
UCT	0.386	0.420	0.432	0.487	0.387
CSGUCT	0.386	0.420	0.432	0.487	0.387
GPBFS	**0.430**	0.449	**0.463**	**0.495**	0.405
LNMCS	0.399	0.449	**0.463**	**0.495**	0.405
NMCS 3	0.405	**0.453**	**0.463**	0.491	**0.407**

Table 6: Final efficiencies found with few algorithms on each graph of size 100 after 600s with greedy playouts

and a greedy algorithm. This result is what pushes us to investigate the use of greedy playouts instead of random playouts.

4.2.3 Greedy playouts and action space reduction

One of the main obstacles we encountered with these experiments is the width of the search tree: with a size of 50, the number of available moves is around a thousand at each state. This poses a problem to the NMCS (which achieves better performance than UCT despite that), as it means billions of score computations. Even LNMCS encounters difficulties in properly evaluating each of these moves when its pruning capabilities help to alleviate this problem.

Inspired by PUCT, which uses a prior neural network to suggest a smaller set of moves when the number of playable moves is too large (like for go with Deepmind's alphago), we make our algorithms only consider the N cheapest moves (here N = 20). Thus, many more expensive moves will not be used, it leads to better results as shown in [2].

Inspired by the good performance of the beam search in tables 4 and 5, even on larger networks, we aim to try greedy playouts on this problem.

We compare the best algorithms with greedy playouts, with action space reduction, and with greedy playouts and action space reduction combined.

The BFS, when using greedy playouts (GPBFS), is slightly modified to swap the node evaluation function to a single greedy playout.

NRPA is by definition applying a learned policy on the playout, replacing it with greedy playouts would turn it into a simple sampling algorithm. The beam search does not use playouts at all. This is why NRPA and beam search are not featured in our experiments involving greedy playouts.

As you can see in Table 6, using greedy playouts alone did not lead to better results. With large graphs and no action space reduction, it is required to compute the efficiency thousands of times per greedy playout, each one requiring applying the Floyd-Warshall algorithm, making each of the playouts very costly. Only a few greedy playouts can be played in 10 minutes (∼15s per greedy playout), which explains the redundancy of the results; LNMCS does not have enough time to prune anything. Both UCT and CSGUCT produced the exact same results because they are very similar. GPBFS is slightly better performing, but is worse than without greedy playouts too. We did not conduct any more experiments on greedy playouts alone because we speculate that the results will be inferior for all algorithms compared to their random playouts results.

In table 7, all algorithms except BFS have better results than in table 5. In addition, it is noticeable that the smaller number of available moves seems to help the NRPA achieve even better results. We suppose it is because the action space becomes small enough for the NRPA to learn a policy. Having a thousand actions available makes it harder to build up the policy. The gap between LNMCS and NMCS 3 results has disappeared compared to the results featured in Table 5. This is unexpected since LNMCS usually performs better than NMCS on most problems. We suppose it might be due to the pruning of the expensive moves: with this setting, all moves lead to good subtrees and LNMCS loses its advantage.

seed	0	1	2	3	4
start value	0.307	0.343	0.357	0.382	0.329
BEAM 10	0.509	0.540	0.564	0.547	0.518
BFS	0.449	0.442	0.462	0.507	0.429
NMCS 3	0.528	0.548	0.555	0.588	**0.529**
LNMCS	0.499	0.545	0.560	0.593	0.522
NRPA	**0.538**	**0.560**	**0.565**	**0.597**	0.514
UCT	0.467	0.497	0.494	0.522	0.465

Table 7: Final efficiencies found with selected algorithms on each graph of size 100 after 600s with action space reduction

seed	0	1	2	3	4
start value	0.307	0.343	0.357	0.382	0.329
GPBFS	0.512	0.568	**0.597**	0.569	0.501
NMCS 3	**0.545**	0.493	0.569	0.537	0.458
LNMCS	0.519	**0.569**	0.592	**0.599**	**0.503**
UCT	0.483	0.502	0.489	0.541	0.470
CSGUCT	0.460	0.511	0.482	0.510	0.470

Table 8: Final efficiencies found with selected algorithms on each graph of size 100 after 600s with action space reduction and greedy playouts

In green, you can see the best results at maximizing the efficiency over all approaches for the graphs of side 100.

The combination of greedy playouts and reduction is shown in Table 8. The results are almost always better than the direct approach, even the previously worse-performing algorithms can outperform the direct approach LNMCS (best algorithm in Table 5). It also slightly outperforms the NRPA in the reduction-only approach except on seed 4. The best algorithm using both action space reduction and greedy playouts is LNMCS, only outperformed by NMCS 3 on seed 0, and by GPBFS on seed 2. GPBFS is a very close second.

GPBFS and LNMCS using greedy playouts and action space reduction, and NRPA using action space reduction, are the three globally dominating approaches to the efficiency problem.

4.2.4 Robustness

Previously, we showed that combining both greedy playouts and action space reduction led to generally better results. This is true for

efficiency, in this sub-subsection we explore a different metric: robustness. Our goal is to determine if the best algorithms are roughly the same with this metric, and how a smaller computational cost of the metric affects the results.

Note that the efficiency used previously is the inverse of the average distance divided by the average distance of the complete version of the graph, according to [4] it is also linked to robustness. Here we need a robustness evaluation of lesser cost, we chose to use the spectral radius: the largest eigenvalue from the adjacency matrix.

seed	0	1	2	3	4
start value	2.512	2.861	3.153	2.738	3.146
BFS	**3.969**	**4.147**	**4.308**	**4.083**	**4.914**
BEAM 10	3.755	3.832	3.943	3.842	4.499
BEAM 100	**3.969**	**4.147**	4.145	**4.083**	4.689
LNMCS	3.728	3.898	3.832	3.932	4.407
NMCS 3	3.857	4.071	4.153	4.045	4.762
NRPA	3.759	3.876	4.045	4.025	4.487
UCT	3.755	3.987	3.488	3.778	3.968

Table 9: Final robustness found for each algorithm on each graph of size 25 after 600s

seed	0	1	2	3	4
start value	3.483	2.755	2.993	3.273	2.706
BFS	4.412	4.895	**4.809**	4.620	4.626
BEAM 10	4.286	4.734	4.383	4.667	5.354
BEAM 100	**5.313**	**5.000**	4.673	**5.097**	**5.782**
LNMCS	4.167	4.869	4.485	4.653	5.116
NMCS 3	4.593	4.880	4.663	4.726	5.114
NRPA	4.049	4.218	4.245	4.228	4.420
UCT	3.989	4.133	4.137	4.215	4.252

Table 10: Final robustness found for each algorithm on each graph of size 50 after 600s

seed	0	1	2	3	4
start value	2.964	3.397	3.085	3.265	3.500
BFS	4.215	5.242	5.071	4.551	5.537
BEAM 10	4.838	**5.467**	**5.144**	**5.627**	**5.633**
BEAM 100	3.788	4.090	4.019	4.164	4.241
LNMCS	**4.988**	5.353	5.134	5.328	5.444
NMCS 3	4.530	5.260	5.110	4.932	5.189
NRPA	3.961	4.222	4.032	4.156	4.303
UCT	3.780	4.111	3.944	4.029	4.145

Table 11: Final robustness found for each algorithm on each graph of size 100 after 600s

seed	0	1	2	3	4
start value	2.964	3.397	3.085	3.265	3.500
UCT	4.508	4.718	4.191	4.358	4.527
CSGUCT	4.508	4.718	4.191	4.358	4.527
NMCS 3	**4.878**	**5.355**	**4.614**	**4.803**	**5.007**
LNMCS	4.775	5.121	4.447	4.764	4.775
GPBFS	4.775	5.121	4.447	4.764	4.775

Table 12: Final robustness found with selected algorithms on each graph of size 100 after 600s with greedy playouts

In green, you can see the best results at maximizing the robustness over all approaches for the graphs of side 100.

Contrary to the good performances of a beam search of width 10 on table 11, using greedy playouts does not allow to beat these results

seed	0	1	2	3	4
start value	2.964	3.397	3.085	3.265	3.500
BEAM 10	4.382	4.783	4.924	4.633	4.592
BFS	4.244	4.514	4.727	4.331	4.757
NMCS 3	**4.633**	**5.214**	**4.981**	4.654	4.674
LNMCS	4.207	4.868	4.974	4.514	4.664
NRPA	4.414	4.724	4.767	**4.825**	**4.856**
UCT	3.918	4.110	4.282	4.057	4.347

Table 13: Final robustness found with selected algorithms on each graph of size 100 after 600s with action space reduction

seed	0	1	2	3	4
start value	2.964	3.397	3.085	3.265	3.500
GPBFS	**4.802**	4.694	5.131	4.546	5.023
NMCS 3	4.743	5.233	4.672	4.593	4.426
LNMCS	4.772	**5.296**	**5.161**	**4.782**	**5.127**
UCT	4.134	4.319	4.477	4.280	4.387
CSGUCT	4.457	4.451	4.684	4.281	4.427

Table 14: Final robustness found with selected algorithms on each graph of size 100 after 600s with action space reduction and greedy playouts

on table 12. The action space is still too large for greedy layouts, only about 20 of them can be played without search space reduction in 10 minutes (30s per playout), which explains the good performances of NMCS over the other algorithms, the playouts are more spread.

The action space reduction does not lead to strictly better results (table 13) than the approach with greedy playouts only, the results are worse than the base approach too. Combined, the two approaches produce slightly better results (table 14) than isolated, but still inferior to the base approach.

Overall, to optimize this definition of the robustness over such graphs, the optimal way seems to use a beam search on the base approach (random playouts, full action space), a good second choice could be the LNMCS on the base approach too.

The goal of this section was not to determine which algorithms provide the best results for this specific robustness metric but to know if we may generalize the results of the various approaches tried on the efficiency over other metrics. We think these results are satisfactory because by trying only one other metric, we can say that the metrics are differently affected by the approaches. We know that greedy playouts and action space reduction are not universally better for this problem depending on the metric we want to optimize.

4.3 Real World Graphs

To better measure our algorithms on the network optimization problem, we decided to apply it to graphs from the Survivable Network Design Library, SNDlib [10]. This library features 26 networks, some represent cities or countries, some are inspired by biology, and some are more abstract. We decided to run the best algorithms from the previous results on 4 graphs from the real world:

1. France, representing the country of France with 25 vertices and 45 edges.
2. Germany50, representing the country of Germany with 50 vertices and 88 edges.
3. India35, representing the country of India with 35 vertices and 80 edges.
4. Cost266, representing the European Union with 37 vertices and 57 edges.

For comparison, Germany50 is the third largest graph in the database, with only the abstract graphs of "brain" and "ta2" bigger with, respectively, 167 and 65 vertices. In Figure 3 you can see what the graphs look like.

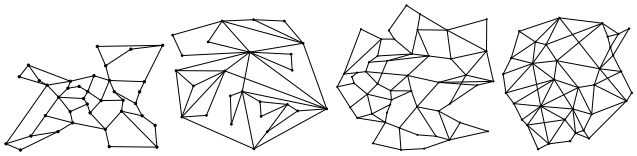

Figure 3: Graphs from SNDlib for Europe, France, Germany, and India from left to right.

Based on the previous results, we selected the best algorithms from each size (25, 50, 100) for robustness (BFS, BEAM 100, BEAM 10) and efficiency (BFS, BEAM 100, LNMCS GR). LNMCS GR is the variant of LNMCS with greedy playouts and action space reductions as it is the one that performs the best overall for optimizing efficiency on large graphs, see table 8. As a baseline, we add the NMCS of level 3 with normal playouts and no action space reduction as it performed well enough in all the experiments.

graph	france	germany50	cost266	india35
start value	0.716590	0.848845	0.844898	0.953861
BFS	**0.884692**	0.883322	**0.912834**	**0.965096**
BEAM 100	0.876070	0.887081	**0.912834**	0.965004
LNMCS GR	0.872731	**0.895397**	0.909709	0.964143
NMCS 3	0.871349	0.885278	0.909772	0.964209

Table 15: Final efficiencies found with selected algorithms on each graph from SNDlib after 600s

graph	france	germany50	cost266	india35
start value	4.712265	4.085959	3.399925	5.491604
BFS	**5.594335**	5.102157	4.760420	**6.354730**
BEAM 100	5.505853	**5.325418**	**4.834799**	6.307945
BEAM 10	5.407345	5.218283	4.494036	6.271883
NMCS 3	5.565474	5.113068	4.799523	6.336223

Table 16: Final robustness found with selected algorithms on each graph from SNDlib after 600s

Except for NMCS 3, the results presented in Tables 15 and 16 are all from deterministic tree search algorithms so they were only executed once.

The results obtained in Tables 16 and 15 corroborate the ones found on synthetic instances: BFS and BEAM give the best results for smaller graphs. Germany50 is the biggest graph and LNMCS already performs well for synthetic instances of size 50, this explains why this graph was best optimized with LNMCS with greedy playouts and action space reduction. NMCS 3 is a control experiment and was not expected to outperform the other algorithms. The graphs from SNDlib have 50 vertices or less and do not give indications of whether the results obtained on synthetic instances apply similarly to graphs with 100 vertices.

5 Conclusion

In this paper, we experimented with optimizing graphs for communications and transport under budget constraints. Multiple different graphs of different sizes were experimented upon, using two definitions of robustness and efficiency among many and with greedy playouts and action space reduction. We showed that while no algorithm is better than the others in any context, the LNMCS usually offers good enough results in most contexts. Deterministic greedy algorithms should not be ignored, as the BEAM search offers great results even on larger graphs. Both UCT and CSGUCT, the previous state-of-the-art algorithms, are outperformed in all situations by all other algorithms tested in this paper. Finally, synthetic graphs seem to behave similarly to real-world graphs, for sizes smaller than 50 vertices at least.

References

[1] Tristan Cazenave, 'Generalized rapid action value estimation', in *24th International conference on artificial intelligence*, pp. 754–760, (2015).

[2] Victor-Alexandru Darvariu, Stephen Hailes, and Mirco Musolesi, 'Planning spatial networks with monte carlo tree search', *Proceedings of the Royal Society A*, **479**(2269), 20220383, (2023).

[3] James E Doran and Donald Michie, 'Experiments with the graph traverser program', *Proceedings of the Royal Society of London. Series A. Mathematical and Physical Sciences*, **294**(1437), 235–259, (1966).

[4] Scott Freitas, Diyi Yang, Srijan Kumar, Hanghang Tong, and Duen Horng Chau, 'Graph vulnerability and robustness: A survey', *IEEE Transactions on Knowledge and Data Engineering*, (2022).

[5] Sylvain Gelly and David Silver, 'Monte-carlo tree search and rapid action value estimation in computer go', *Artificial Intelligence*, **175**(11), 1856–1875, (2011).

[6] Samuel Genheden, Amol Thakkar, Veronika Chadimová, Jean-Louis Reymond, Ola Engkvist, and Esben Bjerrum, 'AiZynthFinder: a fast, robust and flexible open-source software for retrosynthetic planning', *Journal of Cheminformatics*, **12**(1), 70, (December 2020).

[7] Marcus Kaiser and Claus C Hilgetag, 'Spatial growth of real-world networks', *Physical Review E*, **69**(3), 036103, (2004).

[8] Vito Latora and Massimo Marchiori, 'Efficient behavior of small-world networks', *Physical review letters*, **87**(19), 198701, (2001).

[9] Charitha Madapatha, Behrooz Makki, Ajmal Muhammad, Erik Dahlman, Mohamed-Slim Alouini, and Tommy Svensson, 'On topology optimization and routing in integrated access and backhaul networks: A genetic algorithm-based approach', *IEEE Open Journal of the Communications Society*, **2**, 2273–2291, (2021).

[10] Sebastian Orlowski, Roland Wessäly, Michal Pióro, and Artur Tomaszewski, 'Sndlib 1.0—survivable network design library', *Networks: An International Journal*, **55**(3), 276–286, (2010).

[11] Christopher D. Rosin, 'Nested rollout policy adaptation for monte carlo tree search', in *In IJCAI*, pp. 649–654, (2011).

[12] Milo Roucairol., Jérôme Arjonilla., Abdallah Saffidine., and Tristan Cazenave., 'Lazy nested monte carlo search for coalition structure generation', in *Proceedings of the 16th International Conference on Agents and Artificial Intelligence - Volume 2: ICAART*, pp. 58–67. INSTICC, SciTePress, (2024).

[13] Milo Roucairol and Tristan Cazenave, 'Refutation of spectral graph theory conjectures with monte carlo search', in *Computing and Combinatorics*, eds., Yong Zhang, Dongjing Miao, and Rolf Möhring, pp. 162–176, Cham, (2022). Springer International Publishing.

[14] Milo Roucairol and Tristan Cazenave, 'Solving the hydrophobic-polar model with nested monte carlo search', in *International Conference on Computational Collective Intelligence*, pp. 619–631. Springer, (2023).

[15] N. Shanmugasundaram, K. Sushita, S. Pradeep Kumar, and E.N. Ganesh, 'Genetic algorithm-based road network design for optimising the vehicle travel distance', *International Journal of Vehicle Information and Communication Systems*, **4**(4), 344–354, (2019).

[16] D. Silver, Aja Huang, Chris J. Maddison, A. Guez, L. Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Vedavyas Panneershelvam, Marc Lanctot, S. Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and Demis Hassabis, 'Mastering the game of Go with deep neural networks and tree search', *Nature*, **529**, 484–489, (2016).

[17] Maciej Świechowski, Konrad Godlewski, Bartosz Sawicki, and Jacek Mańdziuk, 'Monte carlo tree search: A review of recent modifications and applications', *Artificial Intelligence Review*, **56**(3), 2497–2562, (2023).

[18] Richard Wong, 'A survey of network design problems', (05 2004).