Nested Search for Combinations of Mutations on Boolean Network Ensembles

Xichen Zhang[1], Charles Cazenave[2], Loïc Paulevé[2], and Tristan Cazenave[1]

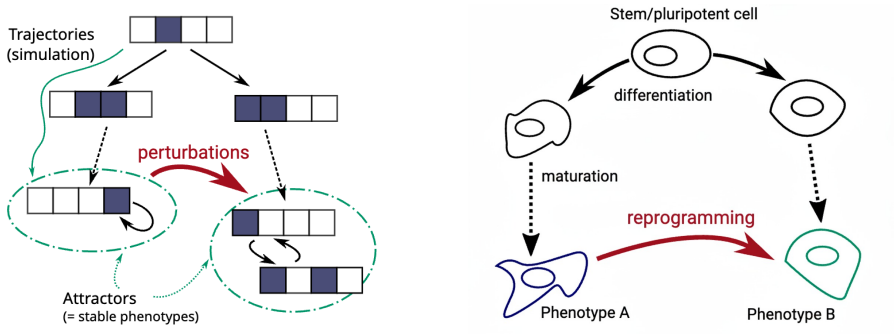
[1] LAMSADE, Université Paris Dauphine - PSL, Paris, France
[2] Univ. Bordeaux, CNRS, Bordeaux INP, LaBRI, UMR 5800, F-33400 Talence, France

Abstract. We study mutation-set search in asynchronous Boolean Network ensembles to control reachable attractors from initial conditions. The robust evaluation of mutations requires averaging over many plausible models. In this setting, each rollout is expensive, while attractor estimates remain noisy under limited simulation budgets. We first derive a variance decomposition that separates across-model diversity from within-model simulation noise, yielding a simple rule for allocating the ensemble size and the number of trajectories. We then compare Lazy Nested Monte Carlo Search (LNMCS) and Nested Rollout Policy Adaptation (NRPA) algorithms. We benchmark the algorithms on different ensemble of Boolean networks coming from biological applications, showing their efficiency on realistic case studies.

1 Introduction

Boolean networks (BNs) are classical models to capture qualitative aspects of gene-regulatory dynamics (see Fig. 1) and cell-fate decisions such as apoptosis, invasion, and metastasis [7]. A BN is a dynamical model in which each node is a binary variable indicating whether it is active (1) or inactive (0), and each node has a logical rule specifying its next state as a function of its regulators. Given an initial configuration, one can simulate the system by applying these rules according to an update mode: in the synchronous mode, all nodes update simultaneously; in the asynchronous mode, a single node updates at a time, chosen non-deterministically. Ultimately, a simulation trajectory reaches a so-called *attractor* of the system. An attractor can either be a single configuration (fixed point), or a set of configurations (cyclic attractor) in which the system can oscillate indefinitely. In biological applications, attractors typically model stable phenotypes of cells, which are often characterized by the value of a subset of nodes, so-called *markers*.

Logical rules of BNs aim to reflect biomolecular regulation, which, in practice, is only partially known. To capture this uncertainty, recent work advocates ensembles of BNs that share structural and dynamical properties but differ in local update rules [17,6,5]. In these works, BN ensembles are the result of BN synthesis algorithms, which derive BN compatible with a given specification, typically derived from a qualitative analysis of experimental data.



(a) Dynamical-model view: discrete states evolve under update rules and settle into attractors.

(b) Cell-fate view: regulatory programs guide differentiation, maturation, and reprogramming decisions.

Fig. 1: Boolean networks as a classical modeling paradigm for regulatory systems.

A central application of BNs in systems biology is the prediction of perturbations to control the (reachable) attractors [21,11]. There is an extensive literature on control and reprogramming methods that, given a single BN, can derive temporary or permanent mutations that modify the reachable attractor landscape towards a desired marker [19,1,13]. In most cases, the main objective is to derive the smallest combinations of mutations to obtain the desired attractors.

Lifted to ensembles, the control problem can be seen with a double optimization objective: on the one hand, finding the smallest combinations (sets) of mutations, and on the other hand, maximizing the probability that the mutations succeed in controlling a BN of the ensemble. The control of BN ensembles has been barely addressed so far. Exact methods such as [20,12] enable identifying combination of mutations that perfectly control all the BNs of the ensemble, which typically necessitate large sets of mutations. In [5], authors used exact control methods on a few BNs of the ensemble, and then evaluate the identified mutations on the full ensemble using stochastic asynchronous simulation with the tool MaBoSS-ensemble. The advantage is that it allows to focus on small sets of mutations with the highest success probability. However, the way the candidate mutation sets are built may preclude discovering better solutions.

In this paper, we evaluate Monte-Carlo based search algorithms to find permanent mutation sets that drive a BN ensemble towards attractors verifying given markers. Throughout, we use asynchronous simulations to approximate steady-state phenotype probabilities, controlled by two parameters: the number of trajectories per model w and the ensemble size M.

The main challenge is to find mutation sets that maximize the steady-state probability of reaching the given marker in the BN ensemble with a limited computation budget, mostly driven by the cost of evaluating a candidate mutation set over the BN ensemble. In the Monte-Carlo Tree Search (MCTS) general framework [8,10,2,9], Nested Monte Carlo Search (NMCS) algorithm specializes

in single-agent optimization by recursively searching for the best sequence [3] and its lazy pruning variant (LNMCS) reduces branching by cheaply screening moves before deeper search [16,15], and Nested Rollout Policy Adaptation (NRPA) adds a learned stochastic rollout policy adapted to the best sequence [14]. We connect these search methods with a simple variance-budgeting analysis for noisy phenotype estimates.

Our contributions are twofold. First, we derive a variance decomposition that separates across-model diversity from within-model simulation noise, which directly yields a practical budget-allocation rule. Second, we provide an implementation of NMCS, LNMCS and NRPA for the control of BN ensemble, and evaluate them on concrete case studies. It results that LNCMS and NRPA are particularly efficient to quickly identify combinations of two to four mutations with high success probability. For double and triple mutations for which the ground truth has been computed, we show that these algorithms are able to identify the optimal solution in a much shorter time than exhaustive screening. Overall, nested search algorithms appear as an effective approach to control BN ensembles.

2 Background and Problem Setup

2.1 Boolean Networks

Let $n \in \mathbb{N}$, $[n] = \{1, \dots, n\}$, and $\mathbb{B} = \{0, 1\}$. A Boolean network (BN) is a function $F : \mathbb{B}^n \to \mathbb{B}^n$,

$$F(x) := \big(f_1(x), \dots, f_n(x)\big) \qquad (x \in \mathbb{B}^n)$$

where $f_i : \mathbb{B}^n \to \mathbb{B}$ is the local Boolean function of nodes $i \in [n]$, and any $x \in \mathbb{B}^n$ is a configuration (see the toy example in Fig. 2). We write $\Delta(x, y) = \{i \in [n] \mid x_i \neq y_i\}$ for the nodes that differ in two configurations. Its *influence graph* $\mathcal{G}(F)$ is a signed digraph over its nodes $[n]$, with $i \xrightarrow{+} j$ (resp. $i \xrightarrow{-} j$) whenever there exist $x, y \in \mathbb{B}^n$ such that $\Delta(x, y) = \{i\}$, $x_i = 0$, and $f_j(x) < f_j(y)$ (resp. $f_j(x) > f_j(y)$).

In this work, we adopt asynchronous semantics to define the transition relation $\xrightarrow[F]{a1}$ between pairs of configurations $x, y \in \mathbb{B}^n$:

$$x \xrightarrow[F]{a1} y \iff \exists i \in [n] : \ \Delta(x, y) = \{i\} \text{ and } y_i = f_i(x),$$

with $\xRightarrow[F]{a1}$ its reflexive-transitive closure. The relation $\xrightarrow[F]{a1}$ forms a digraph over the configuration space $\mathbb{B}^n$. Attractors (including fixed points) summarize long-run behavior under this update mode and correspond to the inclusion-smallest sets of configurations closed by $\xRightarrow[F]{a1}$. A phenotype is a Boolean condition ϕ over configurations.

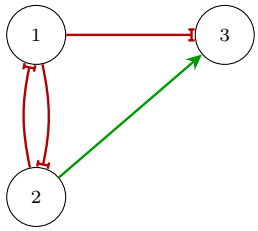


Fig. 2: Influence graph $\mathcal{G}(F)$ for the toy BN. Green: activation; red: inhibition.

Example: toy BN ($n = 3$):

$$f_1(x) = \neg x_2, \qquad f_2(x) = \neg x_1, \qquad f_3(x) = \neg x_1 \wedge x_2.$$

Nodes 1 and 2 form a mutual-inhibition toggle; node 2 activates 3 while node 1 inhibits 3 (Fig. 2). Under asynchronous updates, this yields two fixed points: $(1, 0, 0)$ and $(0, 1, 1)$.

2.2 Mutated Boolean Networks

A mutation fixes a node i to $v \in \mathbb{B}$: (i, v). For a set $\mathcal{M}$ with at most one pair per node,

$$(F/\mathcal{M})_i(x) = \begin{cases} v, & (i, v) \in \mathcal{M}, \\ f_i(x), & \text{otherwise.} \end{cases}$$

Restricting to modifiable regulators $\mathcal{I} \subseteq [n]$ ($g := |\mathcal{I}|$), the k-mutant search space is

$$\mathcal{S}_k = \{\mathcal{M} \subseteq \mathcal{I} \times \mathbb{B} : \ |\mathcal{M}| = k\}, \qquad |\mathcal{S}_k| = \binom{g}{k} 2^k.$$

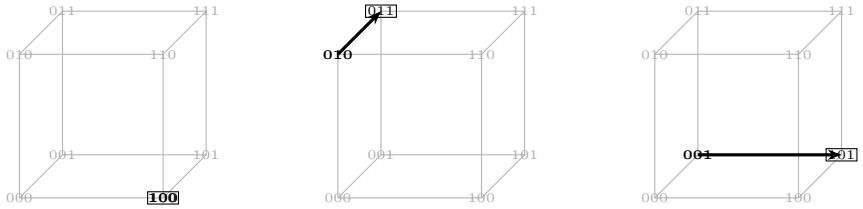
This combinatorial growth motivates the search strategies used later.

Fig. 3 illustrates asynchronous single-step trajectories on the toy BN under single gain-of-function mutations $\mathcal{M} = \{(i, 1)\}$. Each cube displays the configuration graph; thick arrows indicate the enabled update from the annotated start state, and boxed vertices represent fixed points. The three cases show how fixing a single node can reprogram the dynamics toward different attractors: fixing node 1 locks the network at 100, fixing node 2 drives 010 → 011, and fixing node 3 drives 001 → 101.

2.3 Problem Setup

In the following, we consider $\mathscr{F} = \{F_m\}_{m=1}^{M}$, an ensemble of M distinct BNs over the same fixed nodes $[n]$, and an initial configuration $x^0 \in \mathbb{B}^n$.

Given a mutation set $\mathcal{M}$, for a trajectory budget $w \in \mathbb{N}$ per model, for each $m \in [M]$ and each trajectory $j \in [w]$, let $X_m^{(j)} \in \mathbb{B}$ indicate whether a random asynchronous trajectory of length at most n_{steps} in the transition graph generated by $\xrightarrow[F_m/\mathcal{M}]{a1}$, started from x^0, satisfies the phenotype ϕ.



(a) $\mathcal{M} = \{(1,1)\}$, start 100 (fixed). (b) $\mathcal{M} = \{(2,1)\}$, $010 \to 011$ (fixed). (c) $\mathcal{M} = \{(3,1)\}$, $001 \to 101$ (fixed).

Fig. 3: **Asynchronous single-step trajectories on the toy BN (Fig. 2).** Each panel shows the configuration cube under one gain-of-function mutation $\mathcal{M} = \{(i,1)\}$. Thick arrows mark the enabled asynchronous move; boxed states are fixed points.

Evaluation objectives For a mutation set $\mathcal{M}$ and an ensemble $\{F_m\}_{m=1}^{M}$, we estimate the mean success probability $\hat{p}_{\text{ens}}$ as follows:

$$\hat{p}_m = \frac{1}{w}\sum_{j=1}^{w} \mathbb{1}[X_m^{(j)} = 1], \qquad \hat{p}_{\text{ens}} = \frac{1}{M}\sum_{m=1}^{M} \hat{p}_m. \tag{1}$$

Intuitively, M averages over model diversity; w reduces simulation noise per model. Both contribute to cost.

Search problem Given depth D (number of mutations) and a time budget, our goal is to find $\mathcal{M} \in \mathcal{S}_D$ that maximizes $\hat{p}_{\text{ens}}$. Because $|\mathcal{S}_D| = \binom{g}{D}2^D$ grows rapidly, we later employ nested Monte Carlo search with pruning and noise control.

3 Variance Decomposition and Budgeting: Ensemble Diversity Dominates

Given a fixed ensemble size M, how many trajectories w per network are needed to estimate a target-phenotype probability accurately? Larger ensembles average model diversity and typically reduce how large w must be (see Fig. 4).

3.1 Setup

Let $\mathscr{F} = \{F_m\}_{m=1}^{M}$ be a BN ensemble on nodes $[n]$. Fix a simulation protocol (initial state or law, asynchronous updates, and a step cap n_{steps}). For phenotype ϕ, each model F_m has a success probability p_m: under the fixed protocol, it is the probability that an asynchronous trajectory of the mutated model $F_m/\mathcal{M}$ satisfies ϕ before the step cap n_{steps}. Per-model and ensemble estimators $(\hat{p}_m, \hat{p}_{\text{ens}})$ are those defined in Section 2.3 and Eq. (1).

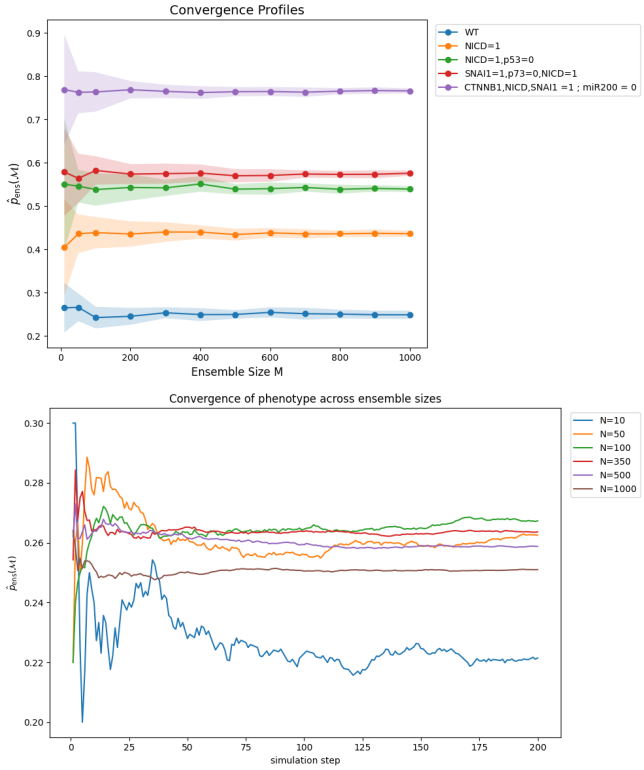


Fig. 4: Convergence under ensemble simulations. Bottom: stability versus trajectories per model w. Top: stability versus ensemble size M.

Assumptions. (i) Within each model, trajectories are i.i.d. given p_m; (ii) different models are independent given $(p_m)_{m=1}^{M}$; (iii) models are sampled uniformly (with replacement) from $\mathscr{F}$.

3.2 Variance decomposition

Lemma 1. *Under the above assumptions,*

$$\mathrm{Var}(\hat{p}_{\mathrm{ens}}) = \frac{\sigma_p^2}{M} + \frac{\eta}{Mw}, \qquad \eta := \mathbb{E}[p_m(1-p_m)], \qquad \sigma_p^2 := \mathrm{Var}(p_m).$$

Proof (Proof sketch). Apply the law of total variance with $\mathrm{Var}(\hat{p}_m \mid p_m) = p_m(1-p_m)/w$ and $\mathbb{E}[\hat{p}_m \mid p_m] = p_m$.

Theorem 1 (Minimal ensemble size). *If* $\mathrm{Var}(\hat{p}_{\mathrm{ens}}) \le \varepsilon^2$, *any* (M, w) *with*

$$M \ge \frac{\sigma_p^2 + \eta/w}{\varepsilon^2}$$

is sufficient. With only $p_m \in [0,1]$ *(hence* $\sigma_p^2 \le \frac{1}{4}$, $\eta \le \frac{1}{4}$*),*

$$M \ge \frac{\frac{1}{4} + \frac{1}{4w}}{\varepsilon^2}.$$

Corollary 1 (Fixed total budget $B = Mw$**).** *With* B *fixed,*

$$M \ge \frac{\sigma_p^2}{\varepsilon^2 - \eta/B} \qquad \textit{(requires } \varepsilon^2 > \eta/B\textit{).}$$

Remark 1 (Simulation-noise-only bound). Since $p(1-p) \le \frac{1}{4}$, enforcing $\eta/(Mw) \le \varepsilon_{\text{sim}}^2$ gives

$$w \ge \frac{1}{4M\varepsilon_{\text{sim}}^2}.$$

The key implication is simple: once w is modest, the σ_p^2/M term from across-model diversity often dominates the $\eta/(Mw)$ term from within-model simulation noise. Under a fixed budget, spending more computation on the ensemble size M is therefore often more beneficial than pushing w much further.

4 Nested Monte Carlo Search for Large Boolean-Network Ensembles

We compare three search variants: NMCS, the lazy baseline LNMCS, and NRPA. All search over mutation sets under the same evaluator, but differ in how they screen candidate moves and how much search guidance they accumulate over time. To make large-ensemble search practical, we use two implementation principles throughout: (i) order-invariant per-level caches for partial mutation sets; and (ii) rollout caching for completed playout evaluations.

4.1 Nested Monte Carlo Search (NMCS)

Nested Monte Carlo Search [3] is a recursive algorithm originally used for single-player games and adapted here to search over mutation sets in BN ensembles. At level 0, it runs a random playout by picking mutations uniformly until the depth limit, returning the resulting score and mutation set. At higher levels, it evaluates each legal move from the current state at one lower nesting level, keeps the best complete mutation set found so far, and then extends the current state with the first unseen move of that best set.

Algorithm 1 mirrors the implementation used in our experiments. States are sorted before caching, so logically identical mutation sets share the same key. The recursive procedure alternates between two phases: evaluating all legal children at level $L-1$, and then expanding the current partial mutation set by the next move suggested by the best returned completion.

4.2 Lazy NMCS (LNMCS)

Lazy nested search (LNMCS) [16,15] keeps the same overall structure as NMCS but adds a depth-dependent pruning rule. Each legal child is first screened by b cheap playouts using the main evaluator. A child whose mean score falls below a threshold θ_d is collapsed to level 0; otherwise it is explored at level $L-1$.

4.3 Nested Rollout Policy Adaptation (NRPA)

NRPA replaces uniform playouts with a learned stochastic policy over mutation choices [14]. In our setting, this provides an alternative form of guidance: rather than pruning branches explicitly, NRPA gradually biases future rollouts toward moves that appear in the best mutation sets found so far.

$$\pi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}, \quad (s,a) \mapsto \pi(s,a),$$

where $\pi(s,a)$ is a log-weight assigned to action $a \in \mathcal{A}(s)$ in state s.

At search level $L = 0$, actions are sampled according to the Gibbs distribution

$$P_\pi(a \mid s) \;=\; \frac{\exp\big(\pi(s,a)\big)}{\sum\limits_{a' \in \mathcal{A}(s)} \exp\big(\pi(s,a')\big)}, \tag{2}$$

which biases selection toward higher-weight actions while retaining stochastic exploration.

After each level-L search, the policy is adapted toward the best sequence $B = (a_1, \ldots, a_D)$ found so far. Let s_t be the state before executing a_t. The update rule is

$$\pi \leftarrow \pi \;+\; \alpha \sum_{t=1}^{D} [e_{\mathrm{code}(s_t,a_t)} - \sum_{a \in \mathcal{A}(s_t)} P_\pi(a \mid s_t)\, e_{\mathrm{code}(s_t,a)}] \tag{3}$$

where e_i is the i-th standard basis vector and $\alpha > 0$ is the learning rate. Equation (3) is a stochastic gradient-ascent step on $\log P_\pi(B)$ under Eq. (2).

5 Experiments

We benchmark the mutation search algorithms presented in the previous section on their performance for identifying combinations of mutations that maximize the reachability of attractors having target markers. Experiments have been performed on two previously published BN ensembles related to the modeling of biological processes (tumour invasion and hematopoiesis). Each algorithm has been given a maximum running time of one hour on a single thread of an Intel(R) Xeon(3) 3.6Ghz CPU.

The implementation and data are available at `https://github.com/Cazou21/Monte-Carlo_Search_for_Boolean_Networks`. Search algorithms and have been

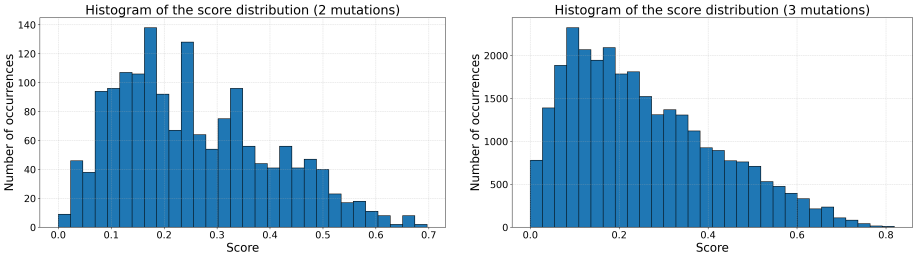


Fig. 5: Distribution of mutation score for all double and triple mutations in case study

implemented in Python, and the asynchronous simulation of BN ensembles has been implemented in Julia, for performance reason. Indeed, the evaluation of a candidate mutation through BN ensemble simulation is the main bottleneck of search algorithms. On the case studies, the Julia implementation enables simulating in average 10-15,000 asynchronous trajectories per second, which is 10-100 times faster simulation compared to equivalent Python implementation, and 3-6 times faster than C++-based MaBoSS depending on the evaluated mutations. As the running time of algorithms is largely dominated by simulation time, the speedup factor directly translates as the speedup of the overall search.

5.1 Case study 1: Tumour invasion BN ensemble

The BN ensemble comprises 1,000 BNs of 32 nodes, all sharing the same influence graph, but having different Boolean functions. The ensemble has been generated by [6] by computing alternative BNs that have the same reachable attractor set than a manually designed BN of tumour invasion. The study [6] has shown that model variability within the ensemble can explain the discrepancies between the base model prediction on control and the experimental data. In this case study, the initial configuration is miR200 = 1, miR203 = 1, and miR34 = 1, while DNAdamage and ECMicroenv are initialized randomly with probability 0.5 each. The target attractor marker corresponds to the phenotype characterized by Invasion = 1, EMT = 1, and Apoptosis = 0.

Ground truth for 2 and 3 mutations We performed the exhaustive evaluation of all possible double and triple mutations on the complete ensemble with 500 trajectories per BN (500,000 trajectories per mutant). This leads to 1,624 double mutations and 29,232 triple mutations. Total computation time was 5 hours for exhausting all double mutations and 49 hours for triple mutations. Fig. 5 show the distribution of score of mutants. The best score for double mutants is 0.70 and 0.82 for triple mutants.

Comparison of NMCS, LNMCS, and NRPA We evaluated the algorithms on the full ensemble with a timeout of 30min to identify the best combination

of two, three, and four mutation. Each algorithm has been run 10 times to account for its variability. Evaluation of candidates was performed using 100 asynchronous simulation per model of the ensemble. The LNMCS algorithm used a lazy evaluation over b=2 playouts (see Fig. 8 for comparison over this parameter). Fig. 6 compares the average, min and max best score obtained along the time with the 10 runs.

In all cases, LNMCS and NRPA clearly outperform NMCS. This can be explained by the large number of playouts that NMCS has to evaluate at each step compared to the two other methods.

For double mutants, the optimal score is found after 260 seconds in most runs of LNMCS and NRPA; 1100 for triple mutations. For quadruple mutations, no ground truth has been computed due to the high number of candidates. LNMCS and NRPA find mutations that outperform triple mutants after minutes.

In average, LNMCS performed 6265 ± 483 evaluations of mutations (consisting of 100 000 simulations) and NRPA 726 ± 312.

5.2 Case study 2: Chevalier BN ensemble of hematopoiesis

The BN ensemble generated by [5] comprises 1,000 BNs of 39 nodes with varying influence graph and initial configurations, but shared properties on the reachable attractors from these initial configurations. The initial configurations and reachable attractors have been designed to match with single-cell transcriptomic data capturing the mouse hematopoietic stem and progenitor cell differentiation. The initial configuration correspond to the stem state, and attractors to differentiated cellular types.

We applied LNMCS algorithm with a timeout of 3600s to identify mutations that would force the system going towards one given attractor. Fig. 7 shows the results of the 10 runs for the reprogramming towards 3 sets of markers defined in [5]: S2 and S4 refers to two distinct differentiated cell type. S2-min and S4-min refers to minimalist markers for these types (4 and 3 nodes respectively), whereas S2-max is a more refined marker for type S2 consisting of the value of 10 nodes. For the min markers, LNMCS consistently reaches the maximum score of 1 even at depth 2. The only exception is the S2-max marker, for which the maximum score is reached at depths 3 and 4, but not at depth 2. Further experiments will evaluate double mutations for reaching each marker.

6 Conclusion

We studied mutation-set search under large Boolean network ensembles, where evaluation cost grows with the number of models and trajectories. The variance decomposition shows that, once a modest number of trajectories is used per model, across-model diversity typically dominates simulation noise. This observation motivates search strategies that preserve search breadth under fixed time budgets. In that regime, state-of-the-art search algorithms LNMCS and NRPA

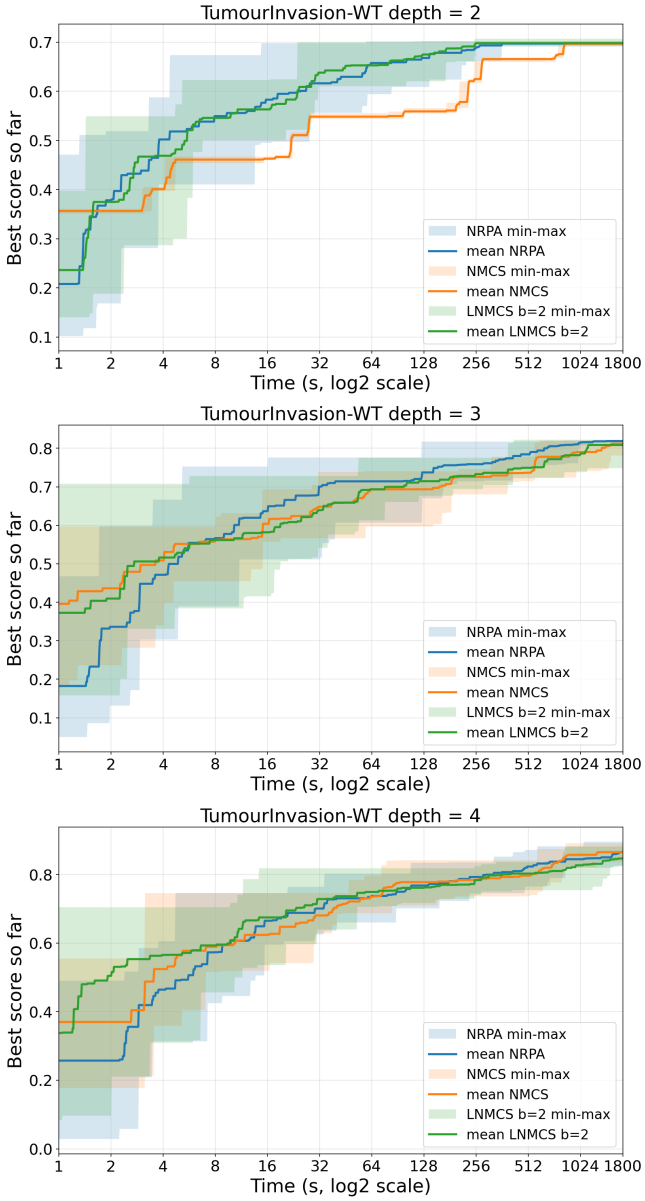


Fig. 6: Pairwise comparison of performances of NMCS, LNMCS and NRPA algorithms on the Case study 1 with different mutation depth (size). For each algorithm, the plots show the average best score of 10 runs along the time, with minimum and maximum best score.

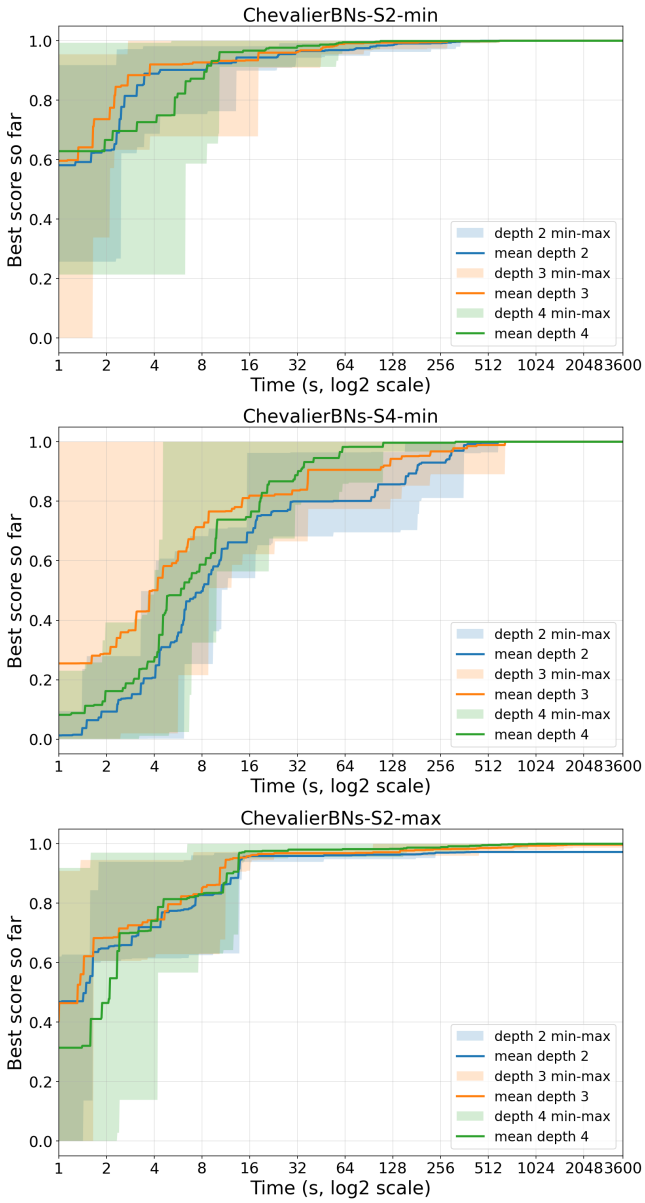


Fig. 7: Average, min, and max best score over 10 runs of LNMCS on the BN ensemble of case study 2 for mutations improving the probability of reaching attractors matching with markers S2-min and S4-min/max.

that we adapted to BN ensemble provide a practical and efficient method for identifying mutation sets that maximize the probability of control success.

As the running time is largely dominated by the evaluation of candidate mutations by simulations, there is a budget-accuracy trade-off. Small $M \times w$ is fast but noisy; large $M \times w$ is accurate but costly. Our current control uses a uniform worst-case bound for the simulation variance, which is conservative. Tighter, data-dependent estimates would better indicate when an estimate is "good enough".

In future work, the tractability of the algorithms should be validated on larger ensembles of larger BNs. As the simulation cost grows, it motivates the design of adaptive simulation strategies, leveraging on the variance decomposition of ensemble size and within-model simulation variability. On the policy-adaptation side, GNRPA generalizes NRPA with temperature and bias terms [4], and recent work has already started to learn these bias weights automatically rather than tune them by hand [18], making the bias-design direction particularly concrete.

References

1. Biane, C., Delaplace, F.: Causal Reasoning on Boolean Control Networks Based on Abduction: Theory and Application to Cancer Drug Discovery. IEEE/ACM Transactions on Computational Biology and Bioinformatics **16**(5), 1574–1585 (Sep 2019). https://doi.org/10.1109/TCBB.2018.2889102
2. Browne, C.B., Powley, E., Whitehouse, D., Lucas, S.M., Cowling, P.I., Rohlfshagen, P., Tavener, S., Perez, D., Samothrakis, S., Colton, S.: A survey of monte carlo tree search methods. IEEE Transactions on Computational Intelligence and AI in Games **4**(1), 1–43 (2012)
3. Cazenave, T.: Nested monte-carlo search. In: Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI). pp. 456–461 (2009)
4. Cazenave, T.: Generalized nested rollout policy adaptation. In: Cazenave, T., Teytaud, O., Winands, M.H.M. (eds.) Monte Carlo Search, Communications in Computer and Information Science, vol. 1379, pp. 71–83. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-89453-5_6, https://doi.org/10.1007/978-3-030-89453-5_6
5. Chevalier, S., Becker, J., Gui, Y., Noël, V., Su, C., Jung, S., Calzone, L., Zinovyev, A., del Sol, A., Pang, J., Sinkkonen, L., Sauter, T., Paulevé, L.: Data-driven inference of Boolean networks from transcriptomes to predict cellular differentiation and reprogramming. npj Systems Biology and Applications **11**, 105 (2025). https://doi.org/10.1038/s41540-025-00569-z
6. Chevalier, S., Noël, V., Calzone, L., Zinovyev, A., Paulevé, L.: Synthesis and Simulation of Ensembles of Boolean Networks for Cell Fate Decision. In: CMSB 2020 - 18th International Conference on Computational Methods in Systems Biology. Lecture Notes in Computer Science, vol. 12314, pp. 193–209. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-60327-4_11
7. Cohen, D.P., Kourou, G., Ioana, I.M., de Jong, H., T., E.S., et al.: Mathematical modelling of molecular pathways enabling tumour cell invasion and migration. PLoS Computational Biology **11**(11), e1004571 (2015), metastasis/invasion Boolean model used in many BN papers

8. Coulom, R.: Efficient selectivity and backup operators in monte-carlo tree search. In: Computers and Games (CG 2006). LNCS, vol. 4630, pp. 72–83. Springer (2007)
9. Gelly, S., Silver, D.: The grand challenge of computer go: Monte carlo tree search and extensions. Communications of the ACM **55**(3), 106–113 (2012)
10. Kocsis, L., Szepesvári, C.: Bandit based monte-carlo planning. In: European Conference on Machine Learning (ECML 2006). LNCS, vol. 4212, pp. 282–293. Springer (2006)
11. Montagud, A., Béal, J., Tobalina, L., Traynard, P., Subramanian, V., Szalai, B., Alföldi, R., Puskás, L., Valencia, A., Barillot, E., Saez-Rodriguez, J., Calzone, L.: Patient-specific boolean models of signalling networks guide personalised treatments. eLife **11** (2022). https://doi.org/10.7554/elife.72626
12. Paulevé, L.: Marker and source-marker reprogramming of Most Permissive Boolean networks and ensembles with BoNesis. Peer Community Journal **3** (2023). https://doi.org/10.24072/pcjournal.255
13. Riva, S., Lagniez, J.M., López, G.M., Paulevé, L.: Tackling Universal Properties of Minimal Trap Spaces of Boolean Networks. In: Pang, J., Niehren, J. (eds.) Computational Methods in Systems Biology. pp. 157–174. Springer Nature Switzerland, Cham (2023). https://doi.org/10.1007/978-3-031-42697-1_11
14. Rosin, C.D.: Nested rollout policy adaptation for monte carlo tree search. In: Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI). pp. 649–654 (2011)
15. Roucairol, M., Arjonilla, J., Saffidine, A., Cazenave, T.: Lazy nested monte carlo search for coalition structure generation. In: Proceedings of the 16th International Conference on Agents and Artificial Intelligence, ICAART 2024, Volume 2, Rome, Italy, February 24-26, 2024. pp. 58–67. SCITEPRESS (2024)
16. Roucairol, M., Cazenave, T.: Solving the hydrophobic-polar model with nested monte carlo search. In: Advances in Computational Collective Intelligence - 15th International Conference, ICCCI 2023, Budapest, Hungary, September 27-29, 2023, Proceedings. Communications in Computer and Information Science, vol. 1864, pp. 619–631. Springer (2023)
17. Schwab, J.D., Ikonomi, N., Werle, S.D., Weidner, F.M., Geiger, H., Kestler, H.A.: Reconstructing boolean network ensembles from single-cell data for unraveling dynamics in the aging of human hematopoietic stem cells. Computational and Structural Biotechnology Journal **19**, 5321–5332 (2021). https://doi.org/https://doi.org/10.1016/j.csbj.2021.09.012
18. Sentuc, J., Ellouze, F., Lucas, J.Y., Cazenave, T.: Learning the bias weights for generalized nested rollout policy adaptation. In: Sellmann, M., Tierney, K. (eds.) Learning and Intelligent Optimization. pp. 194–207. Springer International Publishing, Cham (2023)
19. Su, C., Pang, J.: Cabean 2.0: Efficient and Efficacious Control of Asynchronous Boolean Networks. In: Huisman, M., Păsăreanu, C., Zhan, N. (eds.) Formal Methods. pp. 581–598. Lecture Notes in Computer Science, Springer International Publishing, Cham (2021). https://doi.org/10.1007/978-3-030-90870-6_31
20. Veselý, V., Šmijáková, E., Pastva, S., Beneš, N., Šafránek, D.: Aeon 2025: Robust control of partially-specified boolean networks. In: Fages, F., Pérès, S. (eds.) Computational Methods in Systems Biology. pp. 61–68. Springer Nature Switzerland, Cham (2026)
21. Zañudo, J.G.T., Mao, P., Alcon, C., Kowalski, K., Johnson, G.N., Xu, G., Baselga, J., Scaltriti, M., Letai, A., Montero, J., Albert, R., Wagle, N.: Cell line–specific network models of ER+ breast cancer identify potential PI3kα inhibitor resistance

mechanisms and drug combinations. Cancer Research **81**(17), 4603–4617 (2021). https://doi.org/10.1158/0008-5472.can-21-1208

A Search Algorithms

Algorithm 1: NMCS for BN Ensembles

Input: current mutation set S, level L, depth D, move list $\mathcal{A}$, evaluator ec, optional timeout
Output: best score and mutation set
Function $NMCS(S, L, D, \mathcal{A}, ec, \text{timeout})$
  $T \leftarrow (\text{now} + \text{timeout})$ or $\varnothing$
  $best \leftarrow (-\infty, \emptyset)$; $C \leftarrow [\,\text{Map}()\,]_{\ell=0}^{L}$
  return $\textsc{Core}(S, L, D, \mathcal{A}, ec, C, T, best)$
Function $\textsc{Core}(S, L, D, \mathcal{A}, ec, C, T, best)$
  if $T \neq \varnothing$ ***and*** $\text{now} > T$ **then**
    return $(best.\text{score}, best.\text{set})$
  $S \leftarrow \text{Sort}(S)$; $k \leftarrow \text{Key}(S)$
  if $k \in C[L]$ **then**
    return $C[L][k]$
  if $L = 0$ ***or*** $\textsc{Terminal}(S, D)$ **then**
    if $k \notin C[0]$ **then**
      $C[0][k] \leftarrow \textsc{RandomPlayout}(S, \mathcal{A}, D, ec)$
    $(sc, S^\star) \leftarrow C[0][k]$
    if $sc > best.\text{score}$ **then**
      $best \leftarrow (sc, S^\star)$
    return $(sc, S^\star)$
  $S_{\text{cur}} \leftarrow S$; $bestSc \leftarrow -\infty$
  while ***not*** $\textsc{Terminal}(S_{\text{cur}}, D)$ **do**
    $localSc \leftarrow -\infty$; $localSet \leftarrow \emptyset$
    foreach $m \in \textsc{LegalMoves}(S_{\text{cur}}, \mathcal{A})$ **do**
      $S_1 \leftarrow \text{Sort}(S_{\text{cur}} \cup \{m\})$; $k_1 \leftarrow \text{Key}(S_1)$
      if $k_1 \notin C[L-1]$ **then**
        $C[L-1][k_1] \leftarrow \textsc{Core}(S_1, L-1, D, \mathcal{A}, ec, C, T, best)$
      $(sc, S^\star) \leftarrow C[L-1][k_1]$
      if $sc > localSc$ **then**
        $localSc \leftarrow sc$; $localSet \leftarrow S^\star$
      if $sc > best.\text{score}$ **then**
        $best \leftarrow (sc, S^\star)$
    if $localSc > bestSc$ **then**
      $bestSc \leftarrow localSc$
    $x^\star \leftarrow \text{first}(localSet \setminus S_{\text{cur}})$
    if $x^\star = \varnothing$ **then**
      break
    $S_{\text{cur}} \leftarrow \text{Sort}(S_{\text{cur}} \cup \{x^\star\})$
  return $(bestSc, S_{\text{cur}})$

Algorithm 2: Nested Rollout Policy Adaptation (NRPA)

Input: Level *level*, policy *policy*, number of iterations N
Result: Best score and associated sequence
Function `NRPA`(*level, policy*)
  if $level = 0$ **then**
    return `Playout`(*policy*)
  $bestScore \leftarrow -\infty$
  for $i \leftarrow 1$ **to** N **do**
    $(score, newSeq) \leftarrow$ `NRPA`$(level - 1,$ *policy*`)`
    if $score \geq bestScore$ **then**
      $bestScore \leftarrow score$
      $seq \leftarrow newSeq$
    $policy \leftarrow$ `Adapt`(*policy, seq*)
  return $(bestScore, seq)$
Function `Adapt`(*policy, sequence*)
  $polp \leftarrow policy$
  $state \leftarrow root$
  foreach $b \in sequence$ **do**
    $z \leftarrow 0$
    foreach $m \in$ `Moves`$(state)$ **do**
      $o[m] \leftarrow$ `exp`$(policy[$`Code`$(m)])$
      $z \leftarrow z + o[m]$
    foreach $m \in$ `Moves`$(state)$ **do**
      $p_m \leftarrow \frac{o[m]}{z}$
      $polp[$`Code`$(m)] \leftarrow polp[$`Code`$(m)] - \alpha\left(p_m - \delta_{bm}\right)$
    `Play`(*state, b*)
  return *polp*
Function `Playout`(*policy*)
  $state \leftarrow root$
  while ***true*** **do**
    if `Terminal`(*state*) **then**
      return *(*`Score`(*state*)*,* `Sequence`(*state*)*)*
    $z \leftarrow 0$
    foreach $m \in$ `Moves`$(state)$ **do**
      $o[m] \leftarrow \exp(policy[$`Code`$(m)])$
      $z \leftarrow z + o[m]$
    $move \leftarrow$ `Sample`$\left(m \in \text{`Moves`}(state),\ \frac{o[m]}{z}\right)$
    `Play`(*state, move*)

Algorithm 3: LNMCS for BN Ensembles

Input: current set S, level L, depth D, move list $\mathcal{A}$, evaluator ec, playout count b, pruning ratio r, optional move cap e, optional timeout
Output: best score and mutation set
Function $\textsc{Eval0}(C, S, \mathcal{A}, D, ec, best)$
  $S \leftarrow \mathrm{Sort}(S)$; $k \leftarrow \mathrm{Key}(S)$
  if $k \notin C[0]$ **then**
    $C[0][k] \leftarrow \textsc{RandomPlayout}(S, \mathcal{A}, D, ec)$
  $(sc, S^\star) \leftarrow C[0][k]$
  if $best \neq \varnothing$ ***and*** $sc > best$.score **then**
    $best \leftarrow (sc, S^\star)$
  return $(sc, S^\star)$
Function $\textsc{FastMean}(S, C_f, \mathcal{A}, D, ec, b)$
  return $\frac{1}{b}\sum_{j=1}^{b} \textsc{Eval0}(C_f, S, \mathcal{A}, D, ec, \varnothing)$.score
Function $LNMCS(S, L, D, \mathcal{A}, ec, b, r, e, \text{timeout})$
  initialize deadline T, global best $best$, per-level caches C_m, C_f, and depth statistics $tr, trmax$
  return $\textsc{Core}(S, L, D, \mathcal{A}, ec, C_m, C_f, tr, trmax, b, r, e, T, best)$
Function $\textsc{Core}(S, L, D, \mathcal{A}, ec, C_m, C_f, tr, trmax, b, r, e, T, best)$
  if $T \neq \varnothing$ ***and*** now $> T$ **then**
    return $(best.\text{score}, best.\text{set})$
  $S \leftarrow \mathrm{Sort}(S)$; $k \leftarrow \mathrm{Key}(S)$
  if $k \in C_m[L]$ **then**
    return $C_m[L][k]$
  if $L = 0$ ***or*** $\textsc{Terminal}(S, D)$ **then**
    return $\textsc{Eval0}(C_m, S, \mathcal{A}, D, ec, best)$
  $S_{\text{cur}} \leftarrow S$; $bestSc \leftarrow -\infty$
  while ***not*** $\textsc{Terminal}(S_{\text{cur}}, D)$ **do**
    $\mathcal{M} \leftarrow \textsc{LegalMoves}(S_{\text{cur}}, \mathcal{A})$
    if $e \neq \varnothing$ ***and*** $|\mathcal{M}| > e$ **then**
      $\mathcal{M} \leftarrow \textsc{UniformSample}(\mathcal{M}, e)$
    $d \leftarrow |S_{\text{cur}}|$; $Cand \leftarrow [\,]$
    foreach $m \in \mathcal{M}$ **do**
      $S_1 \leftarrow \mathrm{Sort}(S_{\text{cur}} \cup \{m\})$
      $\bar{s} \leftarrow \textsc{FastMean}(S_1, C_f, \mathcal{A}, D, ec, b)$
      update $tr[d]$ and $trmax[d]$ with $\bar{s}$
      $Cand \leftarrow Cand \cup \{(\bar{s}, S_1)\}$
    $\theta_d \leftarrow tr[d].\text{mean} + r\,(trmax[d] - tr[d].\text{mean})$
    $localSc \leftarrow -\infty$; $localSet \leftarrow \emptyset$
    foreach $(\bar{s}, S_1) \in Cand$ **do**
      $nextL \leftarrow 0$ **if** $\bar{s} < \theta_d$ **else** $L - 1$
      if $nextL = 0$ **then**
        $(sc, S^\star) \leftarrow \textsc{Eval0}(C_m, S_1, \mathcal{A}, D, ec, best)$
      else
        $k_1 \leftarrow \mathrm{Key}(S_1)$
        if $k_1 \notin C_m[L-1]$ **then**
          $C_m[L-1][k_1] \leftarrow$
            $\textsc{Core}(S_1, L-1, D, \mathcal{A}, ec, C_m, C_f, tr, trmax, b, r, e, T, best)$
        $(sc, S^\star) \leftarrow C_m[L-1][k_1]$
      if $sc > localSc$ **then**
        $localSc \leftarrow sc$; $localSet \leftarrow S^\star$
      if $sc > bestSc$ **then**
        $bestSc \leftarrow sc$
      if $sc > best$.score **then**
        $best \leftarrow (sc, S^\star)$
    $x^\star \leftarrow \mathrm{first}(localSet \setminus S_{\text{cur}})$
    if $x^\star = \varnothing$ **then**
      break
    $S_{\text{cur}} \leftarrow \mathrm{Sort}(S_{\text{cur}} \cup \{x^\star\})$
  return $(bestSc, S_{\text{cur}})$

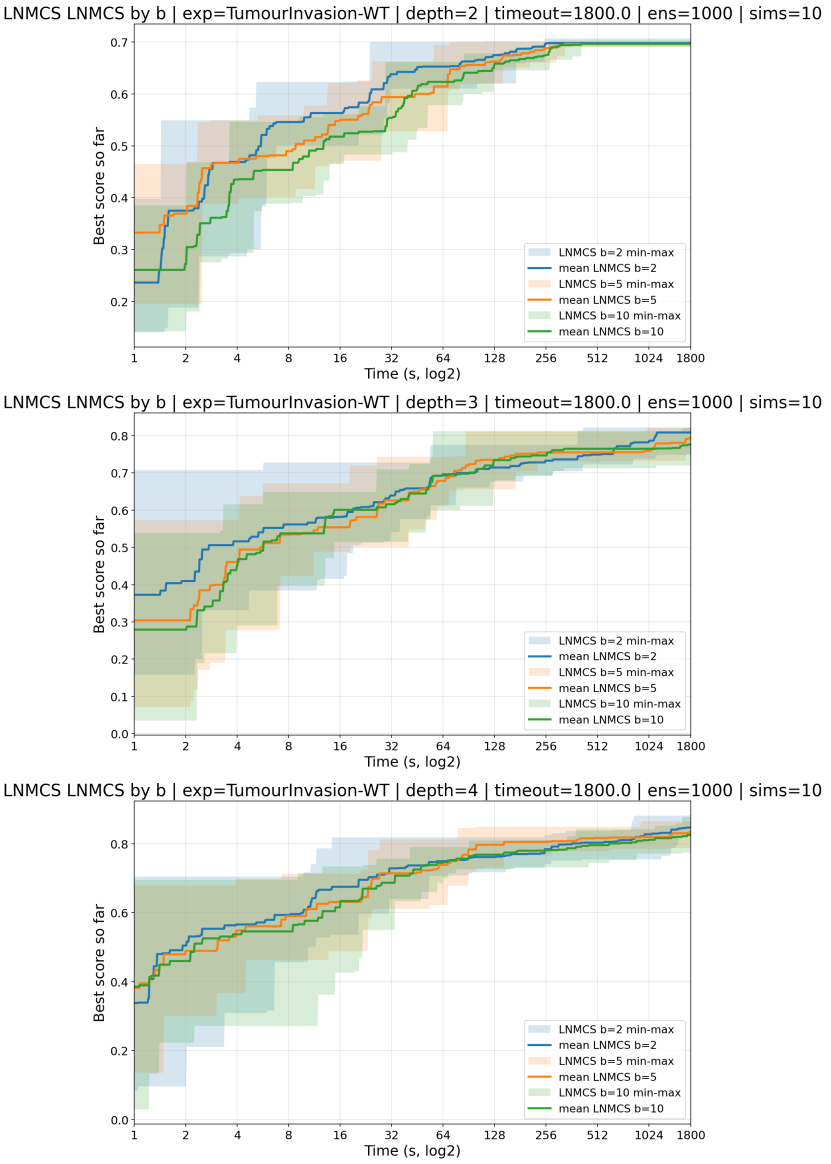


Fig. 8: Average, min, and max best score over 10 runs of LNMCS on the BN ensemble of case study 1 with different parameters b (2, 5, 10)