Improving continuous Monte Carlo Tree Search for identifying parameters in hybrid Gene Regulatory Networks

Romain Michelucci[1], Denis Pallez[1], Tristan Cazenave[2], and Jean-Paul Comet[1]

Université Côte d'Azur, CNRS, I3S, Sophia Antipolis, France
`firstname.name@univ-cotedazur.fr`
LAMSADE, Université Paris Dauphine - PSL, CNRS, Paris, France
`firstname.name@lamsade.dauphine.fr`

Abstract. Monte-Carlo Tree Search (MCTS) is largely responsible for the improvement not only of many computer games, including Go and General Game Playing (GPP), but also of real-world continuous Markov decision process problems. MCTS initially uses the Upper Confidence bounds applied to Trees (UCT), but the Rapid Action Value Estimation (RAVE) heuristic has rapidly taken over in the discrete and continuous domains. Recently, generalized RAVE (GRAVE) outperformed such heuristics in the discrete domain. This paper is concerned with extending the GRAVE heuristic to continuous action and state spaces (cGRAVE). To enhance its performance, we suggest an action decomposition strategy to break down multidimensional actions into multiple unidimensional actions, and we propose a selective policy based on constraints that bias the playouts and select promising actions in the search tree. The approach is experimentally validated on a real-world biological problem: the goal is to identify the continuous parameters of gene regulatory networks (GRNs).

Keywords: MCTS · continuous GRAVE · constraints-based selective policy · action decomposition · chronotherapy · hybrid GRN.

1 Introduction

MCTS is a general decision-time planning algorithm that was initially designed for the improvement of computer Go [13]. The MCTS core idea is to incrementally build a search tree whose nodes represent the states of the environment and edges represent the actions taken from one state to a successor state. MCTS has proved to be effective in a wide variety of settings, including General Game Playing (GGP) [15, 23] but is not limited to games [5, 26]: it can be effective for single-agent sequential decision problems if there is an environment model simple enough for fast multistep simulation. The most popular MCTS algorithm is Upper Confidence bounds applied to Trees (UCT) [19], which addresses the exploration *versus* exploitation trade-off in each state of the tree search using the Upper Confidence Bound [1]. The Rapid Action Value Estimate [16, 17] is a simple yet powerful improvement. The RAVE algorithm combines UCT and the *all-moves-as-first* (AMAF) heuristic [4,6] to provide knowledge sharing between related nodes, resulting in a rapid but biased estimate of the action values. A generalization of the RAVE heuristic [8] has been proposed to gather more accurate estimates near the leaves: the resulting algorithm outperformed RAVE on multiple games such as Go, Atarigo, Knightthrough, and Domineering, without any specific knowledge.

Since the striking success of decision-time planning by MCTS in discrete action spaces, existing methods try to mitigate the requirement of enumerating all actions to deal with large-scale and continuous domains. Progressive Widening (PW) [11,14], also known as progressive unpruning [10], increases the number of child actions of a tree node based on its visitation count. cRAVE [12] adopts the RAVE heuristic using Gaussian convolution-based smoothing, reinforcing information sharing between similar states and actions in each node's sub-tree. Other variants of MCTS in the continuous domain abound. Kernel Regression-UCT [25] generalizes the value estimation between similar actions in a node through kernel regression. Thus, new action generation is guided by kernel density estimation. An alternative to UCT is to replace UCB action selection rule [1] by Hierarchical optimistic optimization (HOO) [7,22] to deal with continuous actions. HOO partitions the action space and builds a binary tree to gradually split it into subspaces. Examples of successful continuous MCTS applications have been: control tasks in OpenAI Gym environment [20], robotic planning [18], and action selection in an Olympic Curling simulator [25].

GRAVE outperformed RAVE in the discrete domain. As far as we know, this paper is the first adaptation of GRAVE to the continuous domain. In addition, we add two generic contributions that improve the MCTS performances in the continuous setting. First, the action decomposition strategy introduced proposes to split a multidimensional action into multiple unidimensional actions. This results in a tree policy reinforcing promising action components instead of the wrong ones and leading to better action selection, specifically when the tree search is shallow. Second, the core idea of a constraints-based selective policy suggests the development of a module that automatically extracts constraints and rules from the domain knowledge to reduce the action space before and during the execution of the procedure.

The paper is organized into three remaining sections: Section 2 presents related works for discrete and continuous MCTS from which we add some contributions detailed in Section 3, and Section 4 provides an experimental study on a real-world biological problem.

2 Monte Carlo Tree Search

2.1 Discrete MCTS

MCTS is a simulation-based tree search in which states of an environment are nodes and actions are edges. The basic version of MCTS uses Monte Carlo simulations for evaluating the nodes of a search tree in order to direct future simulations towards better-rewarded trajectories. Given an initial computation resource (time, memory or number of iterations) and starting at the root node, MCTS executes four steps iteratively:

– selection: a *tree policy* decides which successor node to visit based on a selection function and each successor node statistics,
– expansion: when the selection step ends on a leaf node, the tree is expanded by adding a new node,
– simulation: from the expanded node, a simulation follows a *rollout policy* until a terminal node is reached,
– backpropagation: the reward value of the simulation is assigned to the expanded node and all of its ancestors.

The standard selection function for MCTS is UCB. It follows the principle of *optimism in the face of uncertainty*, which favors the actions with the highest potential value between exploitation (actions with a high mean reward value) and exploration (actions less selected). During the selection step, the action a is chosen (among the set of legal actions $A(s)$ of state s) by applying the UCB formula:

$$argmax_{a \in A(s)}(mean_a + c \times \sqrt{\frac{log(n(s))}{n(s,a)}})$$

where $mean_a$ is the mean reward of a, $n(s)$ is the number of times the state s is selected, $n(s,a)$ is the number of times a is chosen after s is selected, and c is the exploration parameter that must be tuned for each problem: a low value favors exploitation while a high value encourages exploration.

The AMAF heuristic consists of updating the statistics of all actions both selected during the selection and simulation steps. Each of these actions is treated as if it was played on a previous selection step. The reward estimate for an action a from a state s is updated when a is chosen in any playout (even if a is not chosen from s).

RAVE is a popular UCT enhancement that blends the standard UCT score for each node with the AMAF score. Therefore, each node must hold a separate count of rewards and visits for each type. The UCB formula is replaced by:

$$argmax_{a \in A(s)}((1.0 - \beta_a) \times mean_a + \beta_a \times AMAF_a) \qquad (1)$$

where $AMAF_a$ is the AMAF score of the action a, and β_a is the dynamic weight calculated using p_a the number of rollouts starting with a:

$$\beta_a = \frac{pAMAF_a}{pAMAF_a + p_a + bias \times pAMAF_a \times p_a} \qquad (2)$$

in which $pAMAF_a$ is the number of rollouts containing a.

The GRAVE algorithm uses AMAF statistics of an ancestor state if it has more associated rollouts than a given constant called *ref*, which must be tuned for each problem. The idea behind GRAVE is that a state upper in the tree has better accuracy since it has more associated playouts. GRAVE is a generalization of RAVE since GRAVE with *ref* equals zero is RAVE.

2.2 Continuous MCTS

In continuous Markov Decision Processes, standard UCT or RAVE cannot be used since the standard selection step requires the trial of every action at least once, which is impossible in a continuous domain. The progressive widening (PW) heuristic has been proposed to deal with this issue by maintaining a limited number of actions to consider in each state s depending on the number of times s has been visited. Specifically and heuristically, a new child state is sampled from s every time the visitation count of s ($n(s)$) to the power of pw is greater than or equal to its number of children ($n(s)^{pw} \geq |s.children|$). pw is a problem-dependent parameter that controls the number of actions allowed in s. In a nutshell, while UCT ensures that the tree grows deeper in the promising regions of the search space by balancing exploration and exploitation, the PW strategy guarantees that it grows wider in those regions.

cRAVE is an extension of RAVE to the case of continuous action and state spaces. It considers a smooth estimate of action and state values using a Gaussian convolution. Formally, it states that the AMAF score of choosing an action a from a state s is weighted by the contribution related to the state-action pairs (s_i, a_i) encountered in every tree-walk x_s starting from s:

$$AMAF_{s,a} = \sum_{x_s, a_i \in x_s} e^{-log N_{a,s}\{\frac{d(s,s_i)^2}{\alpha_{state}} + \frac{d(a,a_i)^2}{\alpha_{action}}\}} \times \mathbf{R}(x_s) \tag{3}$$

where $\mathbf{R}(x_s)$ is the cumulative reward obtained after following x_s, $N_{a,s}$ denotes the number of state-action pairs involved in every x_s (the sub-tree of s), and α_{action} (resp. α_{state}) is a problem-dependent parameter tuning the importance of $d(a, a_i)$ (resp. $d(s, s_i)$) representing the distance between the action a (resp. state s) and the considered action a_i (resp. state s_i) from the sub-tree. The Euclidean distance is commonly chosen, but the choice of such a measure also depends on the problem. $pAMAF_{s,a}$ is the number of tree-walks containing the state s followed by the action a and is also computed using Gaussian convolutions:

$$pAMAF_{s,a} = \sum_{x_s, a_i \in x_s} e^{-log N_{a,s}\{\frac{d(s,s_i)^2}{\alpha_{state}} + \frac{d(a,a_i)^2}{\alpha_{action}}\}} \tag{4}$$

3 Contributions

Algorithm 1 encompasses the different contributions presented in this section.

3.1 Continuous GRAVE

GRAVE uses AMAF values of a state higher up in the tree than the current state to gather more accurate estimates near the leaves. Its reliability decreases as the number of actions increases: in a continuous action space, the number of times a given action is tried is zero in expectation. An estimation of action and

Algorithm 1 Continuous GRAVE and enhancements

```
Input: $N$ tree-walks, initial state $s_0$, PW parameter $pw$, reference state constant $ref$
Output: A search tree
 1: Initialize constraints from the CSP module
 2: for $i = 1$ to $N$ do
 3:     $s = s_0$, $S = \{s\}$
 4:     while $s$ is not a leaf state and is not simulatable do            ▷ Tree-walk step
 5:         if $n(s)^{pw} < |s.children|$ then                          ▷ PW test, section 2.2
 6:             $sref = s$
 7:             if $n(sref) > ref$ then                          ▷ GRAVE reference state test
 8:                 $sref = s$
 9:             end if
10:             for all $a \in s.children$ do                       ▷ Compute $GRAVE(s,a)$
11:                 $\beta = \frac{sref.pAMAF}{sref.pAMAF + s.p + bias \times sref.pAMAF \times s.p}$        ▷ Eq. 2
12:                 $grave = (1. - \beta) \times s.mean + \beta \times sref.AMAF$          ▷ Eq. 1
13:             end for
14:             Select $a = argmax\{GRAVE(s,a) \mid a \in s.children\}$
15:         else
16:             Sample a new action $a$ from $A_{CSP}(s)$
17:             Add $P(s,a)$ as a child node of $s$       ▷ P(s,a) is the transition function
18:         end if
19:         $s = P(s,a)$, $S = S \cup s$
20:     end while
21:     while $s$ is not a terminal state do                          ▷ Simulation step
22:         Sample $a \in A_{CSP}(s)$ based on default policy
23:         $s = P(s,a)$, $S = S \cup s$
24:     end while
25:     $score = evaluate(s)$
26:     for all $s \in S$ do                                      ▷ Backpropagation step
27:         Update $s$ with $score$                                       ▷ Eq. 3 & 4
28:     end for
29: end for
```

state values must be considered, such as Gaussian convolution smoothing. We propose to adapt the GRAVE algorithm to the continuous domain called *continuous GRAVE* (cGRAVE). The only difference is that the AMAF statistics are updated using Gaussian convolution smoothing (line 27 of Algorithm 1 follows Equation (4)). For computing cGRAVE (lines 10 to 13), the closest ancestor state having more rollouts than a given *ref* constant is kept as a reference state (called *sref* and involved in lines 6-9), and its AMAF statistics are calculated (to calculate lines 11-12).

3.2 Action decomposition

Many real-world problems involve continuous action spaces, $a \in \mathbb{R}^d, d \in \mathbb{N}^*$, where the set of possible actions is not finite. In addition, the dimension d of the action space can also be high, making continuous MCTS methods less effective.

To leverage this issue, we propose an *action decomposition* (AD) strategy, which consists of breaking down each action of d components ($a = (x_1, ..., x_d)$) into d unidimensional actions ($a_1 = x_1, ..., a_d = x_d$). Therefore, the choice of each action component is left to the tree policy. In the expansion step, starting from a state node s, a first action component is sampled. Then, from this action node, a new action component is expanded. By iterating d times, the final component choice leads to a new state node from which the simulation step takes place.
While using this AD strategy, the search tree contains actions from which no simulation can be done. Line 4 of Algorithm 1 ("*not simulatable*") checks whether or not s is a simulatable state. If s is not simulatable, a new action component must be sampled. Finally, if the AD strategy is not chosen, s is always simulatable since its parent action already comprises d components.

The advantage of such a strategy would be that each action node is considered as a traditional tree node in the selection step, leading to a tree policy that gradually reinforces the best action components at the expense of the wrong ones. Figure 1 illustrates such strategy: instead of considering the actions $a_i = (a_{i,1}, ..., a_{i,d})$, $i \in [\![1, n]\!]$ and $n \in \mathbb{N}^*$, as a whole (and the actions resulted by the combination of different components), they are decomposed such that the resulting tree search is composed of the distinct action components.

The effectiveness of this strategy is dependent on the order in which the action components are expanded. In a general framework, it is often impossible to obtain an optimal order of actions, either because the components are dependent or the order function of the components is unknown. If no information is available, a random order of components can be chosen. Otherwise, a heuristic, or even a dynamic calculation of the order of the components, through iterative deepening search, for instance, may reduce the convergence speed and lead to a case of more favorable complexity.

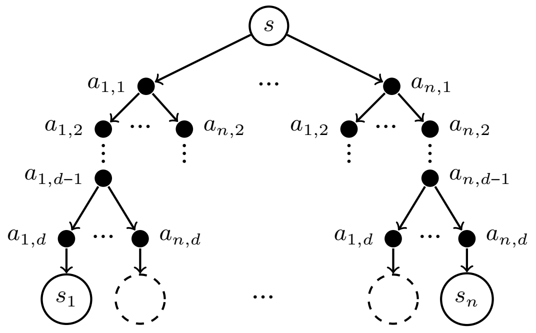


Fig. 1: Action decomposition strategy illustrated (in a deterministic case). The multidimensional actions are decomposed component by component following a particular ordering and forming a tree structure.

3.3 Constraints-based selective policy

The next suggestion we make is a *constraints-based selective policy* (CSP), which takes its roots from selective policies [9], successfully introduced in MCTS variants. The underlying principle is to keep more selectivity in the rollouts. This idea can be applied to any problem by modifying the legal moves set of a node so that moves that are unlikely to be good are pruned during the rollouts. The idea underlying CSP is to automatically extract constraints from the environment definition, some input data, or expert knowledge. The goal is to reduce the action space so that action values that are unlikely to be good or infeasible concerning an extracted constraint are pruned before and during the execution. A module is built to extract constraints that can be used during the expansion step to choose only legal actions and remove ones that will lead to an impossible state (a state in which there is an empty set of legal actions). This module is helpful in the tree to select promising actions, but it can also be during the simulation step, where information sharing (such as kernel regression or AMAF statistics) does not take place: it can be used in playouts to bias the policy.

In real-world applications, the action space $A(s)$ of a state s is bounded, such that each action a_i is defined inside a specified domain D_i. Each domain D_i consists of a set of possible values. The CSP allows extracting a set of constraints C_i, which reduces its corresponding D_i. Such action space is denoted $A_{CSP}(s)$ (lines 16 and 22 in Algorithm 1). These constraints can be implicit or explicit and depend on the application domain. Extracting these constraints helps to reduce the dimension of D_i not only to legal actions but also to actions having a higher probability of being part of a solution. It can be done *a priori* and during the execution of the search. It alleviates the non-locality problem in which some actions may never be reached due to a previously performed action, even if that action belongs to the legal set. The construction of such an extraction module can be viewed as a generalized framework to apply to multiple distinct applications at the price of designing the module. A use case is provided in the next section.

4 Application

We empirically validate our method on the real-world problem of identifying continuous parameters of hybrid gene regulatory networks (hGRNs). First, we describe the problem. Then, the experimental design and setup are detailed. Finally, the results are highlighted and analyzed.

4.1 Parameters identification of hGRNs

Gene regulatory networks (GRNs) modeling is a functional framework for studying and understanding the effect of regulations inside a biological system. Usually, a GRN is represented as a directed graph in which vertices abstract one or multiple biological genes (v_1, v_2 in Figure 2a) and edges express either the

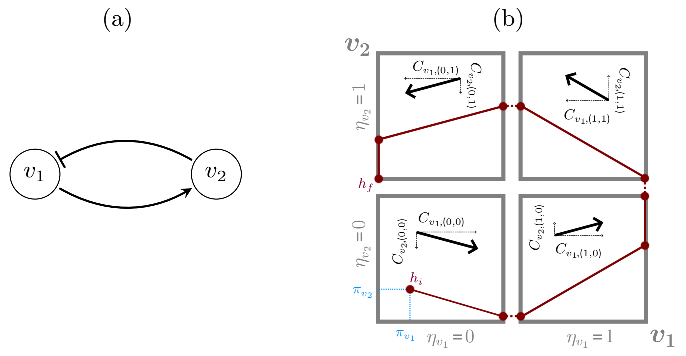
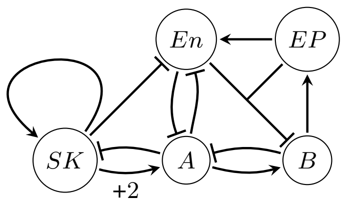


Fig. 2: Example of a hGRN depicted as a directed graph (a), and a possible hybrid state graph (b). The hGRN dynamic parameters are depicted as black arrows.

activation ($\xrightarrow{+n}$) or the inhibition ($\overset{+n}{\dashv}$) of one vertex by another only if the concentration of the source vertex is above its n^{th} threshold. An unlabeled arrow means that $n = 1$. Since each of these regulations may or may not occur, each gene abstraction has a maximum discrete level of concentration corresponding to the maximum number of targets it can regulate. In the context of Figure 2a, the maximum discrete level of both genes is 1, leading to discrete levels of genes (η_{v_1} and η_{v_2}) in $\{0, 1\}$. This graph is a static representation and is of limited interest because it does not help the modeler to predict any evolution of the system. Although a discrete dynamical framework has been developed, we consider here a hybrid modeling framework that adds chronometric aspects to the discrete framework, because it is fundamental to observe and reason not only about the discrete dynamics of a complex system but also about its chronometric evolution. This is particularly important in biology to optimize medical treatments by taking into account biological rhythms (*chronotherapy*). The hGRN dynamics of Figure 2b is then built in two steps: (i) First, each grey box, representing a discrete state $\eta = (\eta_{v_1}, \eta_{v_2})$, defines the discrete concentration level of each gene. (ii) the hGRN dynamics are then defined as piecewise linear continuous trajectories (red lines). The trajectory starts from an initial hybrid state h_i, which is represented by both a discrete state η and a vector determining the precise position π inside the discrete state η. The initial state is defined by $h_i = \left((\eta_{v_1}, \eta_{v_2})^t, (\pi_{v_1}, \pi_{v_2})^t\right) = \left((0,0)^t, (0.25, 0.25)^t\right)$. Then, the evolution inside each discrete state is given by a so-called *celerity vector* (black arrows), which defines the direction and celerity of each gene, e.g., the celerity of v_1 in $\eta = (0, 0)$ is denoted $C_{v_1,(0,0)}$. More generally, the celerity of v in η is denoted $C_{v,\eta}$. Such models could help biologists make new interpretations of the possible system dynamics. Nevertheless, the bottleneck of the modeling framework is the identification of celerity vectors. We are interested in valid hGRN models of the biological system under study, that is, into hGRN models consistent with knowledge and observations.

(a)



(b)

$$\{h_i\}\begin{pmatrix}3.33\\ slide^{-}(EP)\\ SK+\end{pmatrix};\begin{pmatrix}3.33\\ \top\\ SK+\end{pmatrix};\begin{pmatrix}3.33\\ slide^{+}(SK)\\ En-\end{pmatrix};\begin{pmatrix}2.0\\ slide^{-}(En)\\ A+\end{pmatrix};$$

$$\begin{pmatrix}2.0\\ \top\\ SK-\end{pmatrix};\begin{pmatrix}2.0\\ slide^{+}(A)\\ SK-\end{pmatrix};\begin{pmatrix}2.0\\ slide^{-}(SK)\\ B+\end{pmatrix};\begin{pmatrix}2.0\\ \top\\ A-\end{pmatrix};\begin{pmatrix}2.0\\ slide^{+}(B)\\ EP+\end{pmatrix};$$

$$\begin{pmatrix}0.17\\ slide^{+}(EP)\\ En+\end{pmatrix};\begin{pmatrix}0.17\\ slide^{-}(A)\wedge slide^{+}(En)\\ B-\end{pmatrix};\begin{pmatrix}0.17\\ slide^{-}(B)\\ EP-\end{pmatrix}\{h_f\}$$

Fig. 3: Interaction graph of the 5-genes hGRN (a) and its corresponding biological knowledge (b).

Our approach takes into consideration already-formalized information analyzed by biologists derived from biological data and expertise instead of raw data, which are known to be subject to noisiness and scarcity. The approach abstracts the knowledge extracted from biological experiments under the form of constraints on the global trajectory: it must (i) start from an initial hybrid state $h_i = (\eta_i, \pi_i)$, (ii) verify a triplet of properties in each successive discrete state $(\varDelta t, b, e)$ where $\varDelta t$ expresses the time spent; b delineates the observed continuous behavior inside the discrete state ($\top$ means the absence of observed behaviors); e specifies the next discrete state transition, and (iii) reach a final hybrid state $h_f = (\eta_f, \pi_f)$. Figure 3b shows the biological knowledge (BK) associated with the interaction graph Figure 3a of the cell cycle GRN [2] from which we want to extract solutions (valid models) in the next section. Note that the combination of arcs leading from En and EP to B represent a logical conjunction of the two control conditions. Starting from $h_i = \begin{pmatrix}(\eta_{SK}, \eta_A, \eta_B, \eta_{En}, \eta_{EP})^t = (0,0,0,1,0)^t\\ (\pi_{SK}, \pi_A, \pi_B, \pi_{En}, \pi_{EP})^t = (0.5, 0., 0., 1., 1.)^t\end{pmatrix}$, the trajectory must spend 3.33 hours in the discrete state $\eta = (0,0,0,1,0)$. Within this state, the celerity should move towards the next discrete state of SK ($SK+$) to increase the concentration level of gene SK. In the meantime, the trajectory must also reach the minimum admissible concentration of EP ($slide^{-}(EP)$). When SK hits its threshold value, the trajectory jumps into the neighbor state $\eta = (1,0,0,1,0)$. This process continues discrete state after discrete state until the trajectory reaches h_f (= h_i, forming a cycle). Any valuation of dynamic variables, i.e., celerities, leading to a trajectory satisfying this BK, is considered a solution to the hGRN identification problem.

4.2 Decision-making problem specifications

The dimension of the decision-making problem is not the number of genes but a double exponential product: the number of celerities to identify in each discrete state is equal to d (5, here), the number of possible states is $\prod_{v \in V}(b_v + 1)$ with b_v the maximum discrete level of concentration of gene v (48, here), the total number of celerities is $d \times \prod_{v \in V}(b_v + 1)$ (240, here).

To treat this problem with an MCTS approach, we consider each discrete state as an MCTS state and the choice of celerity vectors as an action. In each timestep, an action (the celerity vector choice) is composed of d continuous variables where d is the number of genes. When considering the use case of Figure 3, there are five continuous variables, and, because of biological reasons, the action space domain can be bounded to $[-7.; 7.]^5$. As the BK is based on observations of 12 states, there are 12 continuous multidimensional actions to find a solution leading to a shallow-depth MCTS tree. However, due to the equality constraints on the time criterion in BK, the solution space forms a measure zero set, which complicates the learning process because an action component must find an exact floating-point value. Furthermore, some celerities in one discrete state may be identical in one or more other discrete states, leading to a hard-constrained problem. For this reason, a continuous constraint satisfaction problem solver approach failed to extract even one particular solution when considering these five genes GRN [3]. Finally, the reward for a rollout is the length of the trajectory before it ends in a final state: the maximum is 12 if the trajectory successfully passes between all discrete states (see Figure 3b).

4.3 Design of experiment and experimental setting

The goal of the experiments is to assess the efficiency of the different contributions added to the baseline cRAVE. The design of experiments is cumulative. First, cRAVE is compared to itself combined with the constraints-based selective policy (cRAVE$_{\text{CSP}}$). In the next step, we add to the previous algorithm the action decomposition strategy (cRAVE$_{\text{CSP-AD}}$). Similarly, we replace the cRAVE heuristic in cRAVE$_{\text{CSP}}$ with its improvement GRAVE in the continuous domain (cGRAVE$_{\text{CSP}}$). Finally, we aggregate the different contributions leading to a final version (cGRAVE$_{\text{CSP-AD}}$).

To avoid the disadvantages of the ad-hoc and manual parameter tuning of the algorithms, we decided to use an iterated racing procedure for automatic algorithm configuration. Using the `irace` package [21], we kept the best elite configuration obtained after 1000 iterations to determine the values of the problem-dependent parameters. For the cRAVE heuristics, the parameter space is $bias \in \{1e-15, 1e-14, ..., 1e-2, 0.1\}$, $\alpha_{state} \in [\![1, 100]\!]$, $\alpha_{action} \in [\![1, 100]\!]$, and $pw \in \{0.1, 0.11, ..., 0.89, 0.9\}$. The resulted tuned values are $bias = 0.1$, $\alpha_{state} = 47$, $\alpha_{action} = 87$, and $pw = 0.61$. For the cGRAVE heuristic, ref is found among the values $[\![1, 100]\!]$ and equals 29. The Euclidean distance is chosen for both action and state spaces. Each experiment is run 30 times to obtain statistically significant results. The different policy values obtained are compared for the same computational budget, i.e., 200.000 tree-walks.

4.4 Experiments

Domain knowledge. To develop the CSP module, we used some knowledge about the hybrid framework. Indeed, some celerities, i.e., action components, are constrained to be the same in the different discrete states that will be encountered

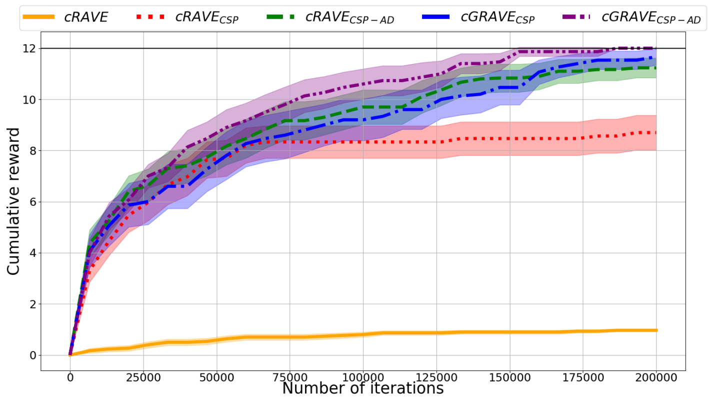


Fig. 4: Comparative performances (cumulative reward) of the different variants on the 5 genes hGRN, versus the computational budget (number of iterations). The upper the better: a reward of 12 means that a solution is found.

by the trajectory. Therefore, *before the execution*, the module propagated some information of BK between the states that share at least one common celerity component. For example, the module automatically extracted that C_{SK} in the third discrete state is constrained by $slide^{+}(SK)$. But it also extracted that the next discrete state shares the same celerity SK. Thus, the module helped to determine that (i) the celerity of SK in the third state is positive due to the $slide^{+}$ knowledge (reducing the search space for this action to only positive values) and (ii) that when the value is known, it is kept to the next discrete state. *During the execution*, the module also helped to evict some action values. In the same example of $slide^{+}(SK)$ in the third discrete state, every positive value for C_{SK} is considered a legal move. However, C_{SK} value is impacted by its entry point, i.e., the hybrid state when entering the third discrete state. As a result, depending on the coordinates of the entry point, C_{SK} values are adjusted online.
There is no domain knowledge about the order function of the moves in the AD, so we have defined an arbitrary order among the genes (first SK, second A, then B, En, and finally EP) and kept the same for every decomposition.

Analysis. Figure 4 comparatively displays the monotonic evolution of the results obtained by the different tested algorithms. It can be observed that (i) the CSP is largely but non-surprisingly beneficial for the convergence of the different MCTS variants: it helps escape an early blockage, (ii) cRAVE$_{\text{CSP-AD}}$ and cGRAVE$_{\text{CSP}}$ both help to improve the findings of better cumulative rewards and (iii) when combining the three contributions, cGRAVE$_{\text{CSP-AD}}$ ensures to always find a solution of the problem, i.e., an admissible valuation of celerity vectors allowing the trajectory to satisfy BK (it hits the maximum threshold near $180,000$ simulations).

Cumulative Distribution Function (CDF) curves are built in Figure 5. Each CDF curve describes the probability of finding a solution at, or below, a given

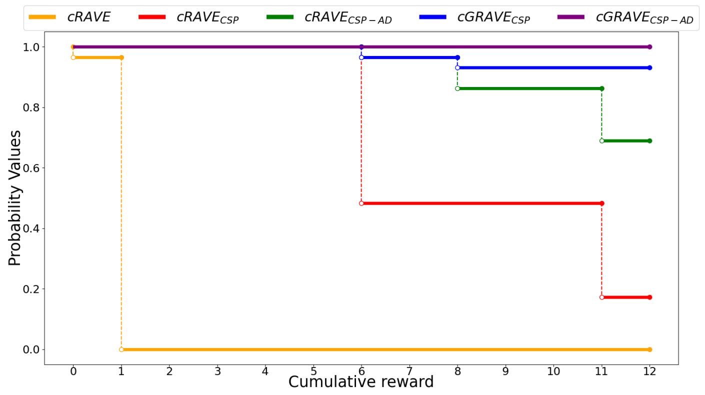


Fig. 5: CDF curves showing the best results for the different variants.

Alg	mean±std	max	min	% of solutions
cRAVE	0.97 ± 0.18	1	0	0
cRAVE$_{\text{CSP}}$	8.7 ± 2.72	**12**	6	20
cRAVE$_{\text{CSP-AD}}$	11.2 ± 1.54	**12**	6	70
cGRAVE$_{\text{CSP}}$	11.6 ± 1.3	**12**	6	93.33
cGRAVE$_{\text{CSP-AD}}$	**12.0 ± 0.0**	**12**	**12**	**100**

Table 1: Statistics of cumulative rewards gathered by the different algorithms tested. Bold values denote the best results column by column.

cumulative reward score. For instance, with cRAVE$_{\text{CSP}}$, there is 100% probability that at the end of a run, a user would obtain a cumulative reward less than or equal to 6, and 50 % probability that the cumulative reward would be less or equal than 11. Table 1 summaries statistics of the best cumulative reward obtained for each run. The mean average and standard deviation of the results are reported, as well as the overall maximum and minimum cumulative reward (the best results column by column are shown in bold). For the maximum, the reader can refer to the x-axis of the rightmost point of each corresponding CDF curve that has more than 0% probability (in the y-axis).

Figure 5 and Table 1 quantitatively illustrate the interest of the different contributions. Without CSP, no solution can be found. And, on top of CSP, the AD strategy and cGRAVE enhance the probability of (i) obtaining a better cumulative reward and (ii) the percentage (%) of problem solutions found. The combination of our proposals (cGRAVE$_{\text{CSP-AD}}$) allows us to obtain a solution with a probability of 100%. Overall, the merits of the contributions are empirically demonstrated in this real-world biological problem.

4.5 Statistical analysis

A statistical validation campaign was conducted to evaluate the observed differences in the reported performance values of all algorithm pairs in order to exhibit

■ Fail to reject H0 □ Reject H0 ($p < 0.05$)

	$cRAVE_{CSP}$	$cRAVE_{CSP\text{-}AD}$	$cGRAVE_{CSP}$	$cGRAVE_{CSP\text{-}AD}$
cRAVE	1.17e-6	6.94e-7	1.45e-7	6.79e-8
$cRAVE_{CSP}$		9.84e-5	5.98e-5	8.89e-6
$cRAVE_{CSP\text{-}AD}$			■ 0.176	6.46e-3
$cGRAVE_{CSP}$				■ 0.179

	$cRAVE_{CSP}$	$cRAVE_{CSP\text{-}AD}$	$cGRAVE_{CSP}$	$cGRAVE_{CSP\text{-}AD}$
cRAVE	1.17e-5	6.94e-6	1.45e-6	6.8e-7
$cRAVE_{CSP}$		9.84e-4	5.98e-4	8.9e-5
$cRAVE_{CSP\text{-}AD}$			■ 1.0	6.46e-2
$cGRAVE_{CSP}$				■ 1.0

Table 2: Pairwise Wilcoxon statistical tests (top) with Bonferroni post-hoc analysis (bottom) for H_0.

the best algorithm variant. We consider the null hypothesis H_0 stating that the observed performance scores are equal. First of all, the choice between parametric and non-parametric tests is made according to the independence of the samples (seeds are different), whether or not the data samples are normally distributed (Kolmogorov-Smirnov test), and the homoscedasticity of the variances (Levene's test). As neither normality nor homoscedasticity conditions required for the application of the parametric tests hold (at $\alpha = 5\%$ confidence level), the non-parametric Friedman rank-sum test is employed to assess whether at least two algorithms exhibit significant differences in the observed performance values. The obtained p-value equals $7e-21$ showing that the differences among the algorithms are significant. However, we still don't know which pairs of algorithms are different. Therefore, the non-parametric Wilcoxon signed-rank test was performed. In a complementary way, to reduce the issue of Type I errors in multiple comparisons, the Bonferroni correction method was applied. Table 2 shows, on top, the p-values obtained with the pairwise Wilcoxon test and, on the bottom, the ones computed with the Bonferroni correction.

If we analyze the conclusions supported by the tests, based on the acceptance or rejection of the above hypotheses, we arrive at the following insights: cRAVE is largely outperformed by the other tested variants. In addition, $cRAVE_{CSP}$ underperforms compared to $cRAVE_{CSP\text{-}AD}$, $cGRAVE_{CSP}$, and $cGRAVE_{CSP\text{-}AD}$. The results achieved by $cGRAVE_{CSP}$ are not statistically different compared to $cRAVE_{CSP\text{-}AD}$ and $cGRAVE_{CSP\text{-}AD}$ (dark boxes in Table 2). However, it can be emphasized that $cRAVE_{CSP}$ lags behind $cGRAVE_{CSP}$ and similarly between $cRAVE_{CSP\text{-}AD}$ and $cGRAVE_{CSP\text{-}AD}$.

4.6 Visualisation

Figure 6 shows the best solution obtained by $cGRAVE_{CSP\text{-}AD}$ for each run. It illustrates the evolution of gene product concentration versus the time spent. The graph type is modeled differently than Figure 2b because of the number of

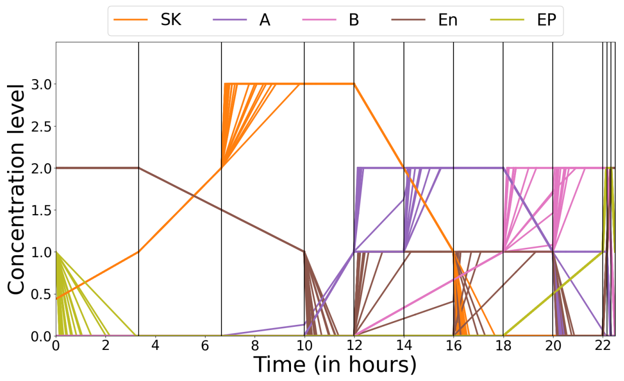


Fig. 6: Visualisation of the 30 solutions (one for each run) obtained by cGRAVE$_{\text{CSP-AD}}$ on the 5 genes hGRN identification problem. Black vertical lines illustrate the 12 different discrete states.

dimensions, but both of them emphasize the same phenomenon: the evolution of concentration (in the y-axis) as a function of the time spent (in x-axis) for the different genes (different curves). This visual confirmation shows that the contributions proposed helped to exhibit solutions, each consistent with BK.

5 Conclusion

The contributions proposed in this work concern first a continuous version of the GRAVE heuristic. GRAVE uses the AMAF statistics of an ancestor node when the number of playouts is too low to have meaningful AMAF statistics on the considered node. The AMAF statistics considered are estimated thanks to a smoothing technique (Gaussian convolutions in our study case) as in the cRAVE approach. In addition, we have presented two additional generic improvements: (i) the action decomposition strategy, which allows having a finer-grained action selection step, and (ii) the constraints-based selective policy implying the construction of a module that automatically extracts constraints to reduce the action space by pruning actions before and during the search process. By analyzing the cumulative experiments on the problem of identifying celerity vectors in a hybrid GRN, the results show that cGRAVE combined with the presented contributions largely outperforms cRAVE. It also emerged that cGRAVE, the action decomposition strategy, and the constraints-based selective policy can independently be generic improvements of MCTS in the continuous domain.

The question of hGRN modeling addressed in this paper actually requires a set of solutions to help the biologist develop new hypotheses and design experiments. This multimodal issue has already been addressed in an alternative approach [24]. We plan to develop a new version of cGRAVE$_{\text{CSP-AD}}$ that can find diverse plans to the same problem in a single run. We believe that such a modification could be useful in other problems, both in the discrete and continuous settings.

References

1. Auer, P., Cesa-Bianchi, N., Fischer, P.: Finite-time analysis of the multiarmed bandit problem. Machine learning pp. 235–256 (2002)
2. Behaegel, J., Comet, J.P., Bernot, G., Cornillon, E., Delaunay, F.: A hybrid model of cell cycle in mammals. In: 6th International Conference on Computational Systems-Biology and Bioinformatics (2015). https://doi.org/10.1142/S0219720016400011
3. Behaegel, J., Comet, J.P., Pelleau, M.: Identification of dynamic parameters for gene networks. In: Proceedings of the 30th IEEE Int. Conf. ICTAI (2018). https://doi.org/10.1109/ICTAI.2018.00028
4. Bouzy, B., Helmstetter, B.: Monte-carlo go developments. Advances in Computer Games: Many Games, Many Challenges pp. 159–174 (2004)
5. 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 pp. 1–43 (2012). https://doi.org/10.1109/TCIAIG.2012.2186810
6. Brügmann, B.: Monte carlo go. Tech. rep., Technical report. (1993)
7. Bubeck, S., Stoltz, G., Szepesvári, C., Munos, R.: Online optimization in x-armed bandits. NIPS (2008)
8. Cazenave, T.: Generalized rapid action value estimation. In: 24th International Joint Conference on Artificial Intelligence. pp. 754–760 (2015)
9. Cazenave, T.: Nested rollout policy adaptation with selective policies. In: Computer Games. pp. 44–56 (2017). https://doi.org/10.1007/978-3-319-57969-6_4
10. Chaslot, G.M.J., Winands, M.H., Herik, H.J.v.d., Uiterwijk, J.W., Bouzy, B.: Progressive strategies for monte-carlo tree search. New Mathematics and Natural Computation pp. 343–357 (2008)
11. Couëtoux, A., Hoock, J.B., Sokolovska, N., Teytaud, O., Bonnard, N.: Continuous upper confidence trees. In: Learning and Intelligent Optimization. pp. 433–445 (2011). https://doi.org/10.1007/978-3-642-25566-3_32
12. Couëtoux, A., Milone, M., Brendel, M., Doghmen, H., Sebag, M., Teytaud, O.: Continuous rapid action value estimates. In: Asian conference on machine learning. pp. 19–31 (2011)
13. Coulom, R.: Efficient selectivity and backup operators in monte-carlo tree search. In: Computers and Games. pp. 72–83 (2006). https://doi.org/10.1007/978-3-540-75538-8_7
14. Coulom, R.: Computing "elo ratings" of move patterns in the game of go. ICGA journal pp. 198–208 (2007)
15. Finnsson, H., Björnsson, Y.: Simulation-based approach to general game playing. In: AAAI. pp. 259–264 (2008)
16. Gelly, S., Silver, D.: Combining online and offline knowledge in uct. In: Proceedings of the 24th International Conference on Machine Learning (2007)
17. Gelly, S., Silver, D.: Monte-carlo tree search and rapid action value estimation in computer go. Artificial Intelligence pp. 1856–1875 (2011)
18. Kim, B., Lee, K., Lim, S., Kaelbling, L., Lozano-Pérez, T.: Monte carlo tree search in continuous spaces using voronoi optimistic optimization with regret bounds. In: Proceedings of the AAAI Conference on Artificial Intelligence. pp. 9916–9924 (2020). https://doi.org/10.1609/aaai.v34i06.6546
19. Kocsis, L., Szepesvári, C.: Bandit based monte-carlo planning. In: Fürnkranz, J., Scheffer, T., Spiliopoulou, M. (eds.) Machine Learning: ECML. pp. 282–293 (2006). https://doi.org/10.1007/11871842_29

20. Lee, J., Jeon, W., Kim, G.H., Kim, K.E.: Monte-carlo tree search in continuous action spaces with value gradients. In: Proceedings of the AAAI conference on artificial intelligence. pp. 4561–4568 (2020). https://doi.org/10.1609/aaai.v34i04.5885
21. López-Ibáñez, M., Dubois-Lacoste, J., Pérez Cáceres, L., Stützle, T., Birattari, M.: The irace package: Iterated racing for automatic algorithm configuration. Operations Research Perspectives pp. 43–58 (2016). https://doi.org/10.1016/j.orp.2016.09.002
22. Mansley, C., Weinstein, A., Littman, M.: Sample-based planning for continuous action markov decision processes. In: Proceedings of the International Conference on Automated Planning and Scheduling. pp. 335–338 (2011)
23. Méhat, J., Cazenave, T.: A parallel general game player. KI-künstliche Intelligenz pp. 43–47 (2011). https://doi.org/10.1007/s13218-010-0083-6
24. Michelucci, R., Callegari, V., Comet, J.P., Pallez, D.: Cellular genetic algorithms for identifying variables in hybrid gene regulatory networks. In: Applications of Evolutionary Computation (EvoApplications) (2024). https://doi.org/10.1007/978-3-031-56852-7_9
25. Yee, T., Lisỳ, V., Bowling, M.H., Kambhampati, S.: Monte carlo tree search in continuous action spaces with execution uncertainty. In: Proceedings of the 25th IJCAI. pp. 690–697 (2016)
26. Świechowski, M., Godlewski, K., Sawicki, B., Mańdziuk, J.: Monte carlo tree search: a review of recent modifications and applications. Artificial Intelligence Review (2023). https://doi.org/10.1007/s10462-022-10228-y