



# Monte Carlo Tree Search as an intelligent search tool in structural design problems

Leonardo Rossi<sup>1</sup> · Mark H. M. Winands<sup>2</sup> · Christoph Butenweg<sup>3</sup>

Received: 12 October 2020 / Accepted: 5 February 2021 / Published online: 26 February 2021  
© The Author(s) 2021

## Abstract

Monte Carlo Tree Search (MCTS) is a search technique that in the last decade emerged as a major breakthrough for Artificial Intelligence applications regarding board- and video-games. In 2016, AlphaGo, an MCTS-based software agent, outperformed the human world champion of the board game Go. This game was for long considered almost infeasible for machines, due to its immense search space and the need for a long-term strategy. Since this historical success, MCTS is considered as an effective new approach for many other scientific and technical problems. Interestingly, civil structural engineering, as a discipline, offers many tasks whose solution may benefit from intelligent search and in particular from adopting MCTS as a search tool. In this work, we show how MCTS can be adapted to search for suitable solutions of a structural engineering design problem. The problem consists of choosing the load-bearing elements in a reference reinforced concrete structure, so to achieve a set of specific dynamic characteristics. In the paper, we report the results obtained by applying both a plain and a hybrid version of single-agent MCTS. The hybrid approach consists of an integration of both MCTS and classic Genetic Algorithm (GA), the latter also serving as a term of comparison for the results. The study's outcomes may open new perspectives for the adoption of MCTS as a design tool for civil engineers.

**Keywords** Monte Carlo Tree Search · Structural design · Artificial intelligence · Civil engineering · Genetic algorithm · Reinforced concrete buildings

## 1 Introduction and state of the art

Modern structural engineering is a discipline that strongly relies on computationally intensive methods. On the one hand, procedural, rule-based work-tasks as calculating,

drawing, and preparing technical reports are already largely automated and integrated in Building Information Modelling (BIM), also thanks to widespread commercial and scientific applications. On the other hand, for those tasks concerning higher-level skills—as choosing an effective load-bearing structural system, construction material or supporting the idea of automated design—there is plenty of space for research, especially within the fiercely expanding area of Artificial Intelligence (AI) [1]. The application of AI in structural engineering has the potential to improve the work efficiency of experienced engineers, to support effectively unexperienced technicians and to share and spread the knowledge within professional engineering teams. First successful attempts to use AI and computers for the generation of design concepts date back to the 1980s (e.g. [2, 3]). Since then, several projects focusing on various techniques as inductive learning (e.g. [4, 5]), deep learning and neural networks (e.g. [6–9]), constraint search (e.g. [10]), evolutionary computation and genetic algorithm (e.g. [11–14]), cellular automata (e.g. [15, 16]) or generative approaches (e.g. [17]) were carried out.

<sup>1</sup> Department of Engineering, Università degli Studi di Perugia, via Goffredo Duranti 93, 06125 Perugia, Italy

<sup>2</sup> Department of Data Science and Knowledge Engineering (DKE), Faculty of Science and Engineering, Maastricht University, P.O. Box 616, 6200 MD Maastricht, the Netherlands

<sup>3</sup> Center for Wind and Earthquake Engineering (CWE), RWTH Aachen University, Mies-van-der-Rohe-Straße 1, 52074 Aachen, Germany

Looking at existing studies, we found that, while developing an intelligent tool to solve structural design tasks, limitations may arise with regards to, at least, three different aspects: (1) the lack of generality and portability of the selected methodologies; (2) the unavailability of sufficient training data; (3) a computational effort that grows too rapidly as problems' complexity grows. Considering this, we believe that a successful approach to the problem of automated structural design should be (1) as portable as possible, (2) data independent, and (3) computationally efficient. Interestingly, in 2016, Silver et al. [18] showed that adopting the so-called Monte Carlo Tree Search (MCTS) algorithm [19]—with support of deep neural networks [20]—can represent a huge step forward in this sense. In particular, Silver et al. [18] demonstrated how computers can outperform humans on the complex board game Go [21]; such task was for long considered untreatable by machines, as the game has a search space of  $10^{172}$  possible different configurations, and requires the development of a long-term game strategy. Since its inception, MCTS was indeed successfully implemented in many board games [22–25], protein folding problems [26], chemical design applications [27], planning and logistics [28, 29], and is currently intended as one of the best approaches in Artificial General Intelligence for games (AGI) [30]. Interestingly, as of 2021, only a small number of engineering-related applications of MCTS exist [31, 32].

As discussed in the next section, MCTS has the potential to be successfully applied to many different civil engineering problems, especially those concerning structural design. Indeed, on the one hand, the latter is characterised by rules and procedures that can easily be translated into code. On the other hand, complex design processes in engineering are often solved on the basis of experience and already known solutions from previous projects. In many cases, this leads to uneconomic structural solutions, as the limited view in combination with working time pressure prevents innovative and problem-oriented alternatives. The support of complex design processes in civil engineering may require evaluating a huge number of different solutions. Interestingly—as observed by Edelkamp et al. [29]—MCTS can deal not just with “*deterministic, fully observable, discrete, static*” environments, but also with “*stochastic, partially observable, dynamic, continuous*” environments. The development of structural design and structural safety procedures may benefit from this latter point.

In the following, first of all, we introduce MCTS. Secondly, we report the pseudocodes adopted for the considered application. As a third step, we describe the engineering problem we focused on, also providing insights about how to actually implement the methodology in the specific engineering context. Fourth, we describe a case study, so to provide the reader with numbers about MCTS's performance as a search tool. Finally, we evaluate results

and then discuss possible limitations, also taking Genetic Algorithm [33] as a term of comparison.

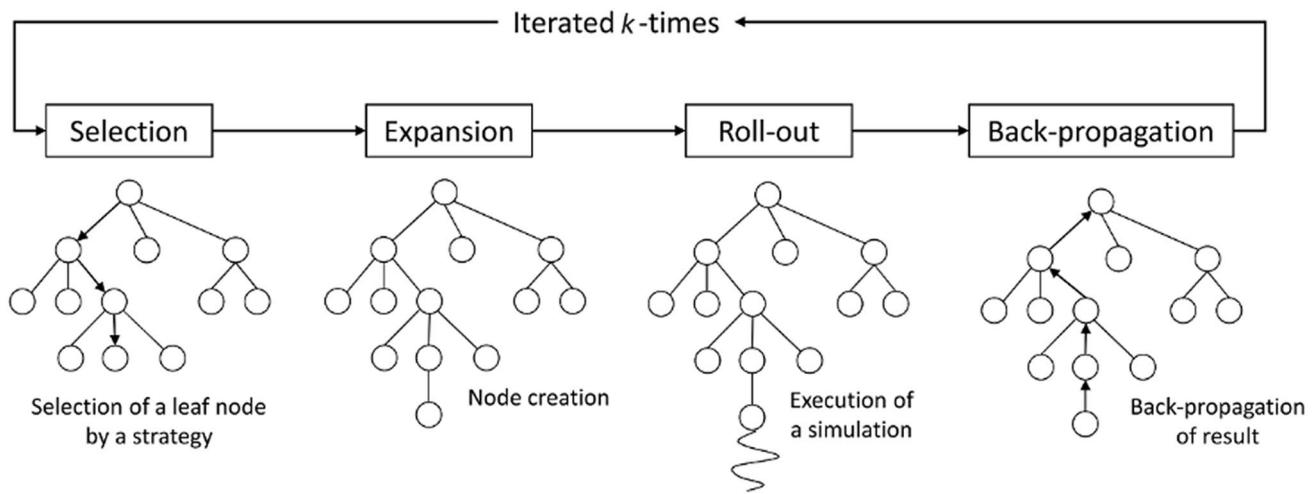
## 2 Methodology

### 2.1 Monte Carlo Tree Search

MCTS (e.g. [19, 34]) is a search technique that works on a tree data structure; from a single initial node (the root), other nodes are added that represent different instances of the search space  $S$ . The MCTS is *aheuristic*, i.e. it does not require task-dependent heuristics [35]: knowing the available actions at each step, and their corresponding outcomes (including stop conditions) is sufficient for it to run. Also, the algorithm is *anytime*, i.e. it can determine the next step to take within any time budget. Finally, it is *selective*, that means it can grow the search tree asymmetrically, favouring those branches that host the most promising moves [36]. MCTS is often used in board- and video-game environments [25, 30, 37–39], where a tree's node may represent a player's move, while the end of a match stops the expansion of the tree.

MCTS algorithm consists of four main steps, i.e. *Selection*, *Expansion*, *Roll-out* and *Back-propagation* (see Fig. 1—adapted from [35]). The steps are repeated iteratively until a stop condition emerges. During *Selection*, starting from the root, the tree is descended so as to find a not fully expanded node— $n^*$ —for the sake of brevity, here referred to as a *leaf*. The path from the root to the leaf is determined by using a selection policy (e.g. [40]) that suggests the best promising moves. Then, from the selected leaf, a node is added to the tree (*Expansion* phase). The branch to be expanded can be selected randomly, while avoiding the repetition of already existing sibling nodes. As a third step, the *roll-out* (i.e. a simulation that continues until a terminal state is reached) is performed; during it, nodes are generated and temporarily incorporated into a solution, but not permanently added to the tree. Finally, the reward of the roll-out is *back-propagated* from the node selected for expansion ( $n^*$ ), back to the tree's root. In this way, the agent is informed about the quality of the search iteration it performed. In Fig. 2, we show MCTS's pseudo-code, adapted to our study from existing literature (e.g. [30]), is shown. As a first research step, we adopted the algorithm's basic formulation; other, more sophisticated approaches, such as SP-MCTS [41] may be considered in further research initiatives. In the present work, both the algorithmic and the engineering parts were coded from scratch, in Java.

The meaning of the parameters included in Fig. 2 are given in the following.



**Fig. 1** A scheme of the Monte Carlo Tree Search algorithm—[35] adapted from Chaslot et al.

- **PD, problem's dimension.** It is the number of single different moves the MCTS agent can take as a first step (i.e. the *branching factor* at root node).
- **FG, fitness goal.** It is a real number in the range [0, 1], representing the performance goals in terms of solution's fitness.
- **RL, roll-out length.** It is the maximum number of moves to be played on the roll-out phase. We set it as a random integer between 1 and the number of moves actually possible at the considered step. Randomness is adopted as we want not all the roll-outs to end with the same number of played moves. The point is, in our non-adversarial (1-player) approach, there is no direct opponent that can terminate the descent of the tree by winning/losing/drawing the game; we wanted to avoid that roll-outs not reaching the performance goal (a stop condition) would finish with a saturated board, as this would have biased the search.
- **move\_available (n).** It is a function that returns *true* if the tree can be expanded; it returns *false* otherwise.
- **is\_leaf (node).** It is a function that, given a node, returns a Boolean value about it being or not being a leaf node.
- **best\_child (node).** It is a function that, for any given node, returns its best performing child-node (or one of its best child-nodes, in case of an *ex aequo*). As shown between lines 7 and 9 of the pseudo-code of Fig. 2, the MCTS algorithm calls *best\_child* function as soon as the tree descent reaches a node that is not a leaf.
- **add\_random\_child\_node (node).** It is a function that, given a node, adds randomly one of the non-explored successor nodes.
- **roll\_out (node, int).** It is a function that, from the given node and *max number of moves*, performs roll-out and returns the level of fitness performed.

- **back\_propagate (node, double).** It is a function that takes as an input both a node and a value of fitness, and back-propagates the latter from the given node, back to the root of the tree.

The back-propagation consists of updating both the sum of rewards (expressed in terms of fitness value) and the number of simulations executed, for each node between the tree's root and the node from which the last roll-out was launched. In the following algorithm's iteration (if any), such parameters are used by the *best\_child* function, to determine the most promising node so far. In particular, in this study, we follow the *Upper Confidence Bound 1 applied to Trees* (UTC) approach (see Formula 1), as suggested by Kocsis and Szepesvári [40] to balance both *exploration* and *exploitation* of the state under consideration:

$$\operatorname{argmax}_i \left\{ \frac{\sum r_i}{n_i} + c \cdot \sqrt{\frac{\ln N_i}{n_i}} \right\}. \quad (1)$$

The meaning of the literals of Formula 1 is the following:  $i$  is the index of the node,  $\sum r_i$  is the sum of the rewards for child node  $i$ ,  $n_i$  is the number of simulations conducted for child node  $i$ ,  $N_i$  is the total number of simulations executed from the parent node of node  $i$ ,  $c$  is the *exploration parameter*, empirically set. In our study we fine-tuned the value to 0.8.

While solving a search task, MCTS does not carry out a full simulation of the search space. On the contrary, the search tree usually grows in an asymmetric fashion.

---

**Monte Carlo Tree Search Algorithm**


---

**Input:**

- PD: problem's dimension
- FG: fitness goal
- RL: roll-out length

**Output:**

- the best solution found,  $n^*$
- the maximum performance level reached,  $f$
- the total number of roll-outs performed,  $t$

```

1  create root node  $n_r$  and add it to the tree
2  make  $n_r$  actual node  $n$ 
3   $n^* = \text{null}$ 
4   $f = 0$ 
5   $t = 0$ 
6  while(move_available( $n$ ))
      // descend the tree to a Leaf
7    while(!is_leaf( $n$ ))
8       $n = \text{best\_child}(n)$ 
9    end
10    $n = \text{add\_random\_child\_node}(n)$ 
11   RL = rand(PD,  $n$ )
12    $f, n' = \text{roll\_out}(n, RL)$ 
13    $t++$ 
14   back_propagate( $n, f$ )
15   if( $f > FG$ )
16      $n^* = n'$ 
17     break
18   end
19    $n = n_r$ 
20 end
21 return  $n^*, f, t$ 

```

---

**Fig. 2** MCTS' pseudo-code adapted in the present study

## 2.2 Adapting the reference algorithm to the envisaged problem

In our study, we adapted the default MCTS methodology to the specific needs of the envisaged engineering problem, as follows. First of all, the search is intended at finding reinforced concrete (RC) structures that satisfy a given *fitness goal* in terms of specific engineering aspects. We intend a structure as made of a rigid slab (that incorporates rigid horizontal RC beams), supported by vertical RC bearing elements (columns and walls); the algorithm creates different design hypotheses (or trials) by changing (1) shape, (2) size, (3) number and (4) position of such elements. In this context, the *root node* represents an empty design, while any other *node* represents a trial RC structure with certain

characteristics and corresponding fitness value. The *tree* is the data structure in which all nodes are organized. In Go-related applications of MCTS (e.g. [39]), adding one node to the tree means placing one stone on the game's board. In our study, the addition of a node to the search tree represents the insertion of a unitary bearing element (i.e. the minimum amount of structural material that can be added, in the following also called *design unit*) to an existing hypothesis. In this way, the algorithm creates *child-nodes* from nodes; the existence of a logical edge between parent- and child-node makes it possible to link a trial structure's performance to that of all its ancestors.

A node is called a *leaf* as long as child-nodes can be added to it (independently of its position within the tree). In the practical implementation of MCTS we propose, there is

a limit to the maximum number of child-nodes (as discussed in Sect. 4). For what concerns *tree descent*, in this context it means going from the empty solution down to a leaf node, where a new trial structure can be added. As a trial structure (a node) is added to the tree, its potential as a possible intermediate step towards the final structural design is evaluated by executing a *roll-out*. This consists of adding (without including the corresponding nodes into the tree) a random amount of bearing elements to the above-mentioned trial and then testing the solution's fitness. After this step, the performance of all the intermediate trial structures between the one from which roll-out started and the tree's root is updated by *back-propagation*.

In two-player game applications, as presented by Silver et al. [39], before every move, each player performs one or more roll-outs (according to a search budget), to be able to identify the best next action. In general, the match's outcome after a move cannot be controlled by the agent, as it is not known in advance which decision the opponent will take after it. For this reason, a high ratio of won games has to be discovered along the tree, and used to calculate chances of success for the moves to take. In our case study, there is no opponent for the agent; so (1) every path leading to a sufficiently performing solution corresponds to reaching the goal [41], and (2) any path of the search tree, in accordance with the budget of moves, is de facto possible. Considering this, and considering that we express performance as a decimal number in the range [0, 1], in order to provide feedback to the algorithm, we took the level of performance achieved at the end of a path as a proxy for  $r_i$ . The measure we adopted for performance is presented in Sect. 3.

### 2.3 Term of comparison

In this work, we consider a Genetic Algorithm (GA) [33, 42, 43] as a benchmark for the results obtained with MCTS. The GA-approach was chosen as it is largely widespread in multi-objective search problems [44]; also, it has some relevant features similar to MCTS. First of all, it does not need an analytical model of the system to be investigated, but rather just a list of admissible actions and corresponding consequences at each step. Second, it can start from zero knowledge about the quality of the solutions; in other words, no external data are needed to train the algorithm. Third, it can be interrupted at any time while running: stopping it before reaching the performance trigger would just affect the goodness of the obtained solution. Fourth, both MCTS and GA can explore a search space in an asymmetrical way. This is important for civil engineering applications, where the number of branches of technically possible solutions is extremely large, but most of them have to be disregarded as just tiny minority scores sufficiently well. As a parallel

to Fig. 2, the pseudo-code of the GA algorithm we implemented (in Java) is given in Fig. 3.

The meaning of the parameters listed in Fig. 3 is given in the following:

- FG, *fitness goal*. It is a real number within the range [0, 1], representing the performance goals in terms of solution's fitness.
- PS, *population size*. It is the number of solutions that form each generation.
- NI, *maximum number of iterations*. It is the maximum number of generations that can be produced by the agent during the search.
- CR, *crossover rate*. It is the percentage of solutions selected for reproduction by means of crossover.
- MR, *mutation rate*. It is the percentage of solutions undergoing mutation. In a given solution, a mutation consists of losing and/or getting a number of cells between 1 and 5.
- IR, *immigration rate*. It is the percentage of brand-new solutions introduced at each generation.

Numerical values of the parameters are reported in Sect. 4.

### 2.4 The agents

In the following, we show two different agents based on Monte Carlo Tree Search: the first one (also called MCTS) is a direct implementation of a plain, 1-player version of the algorithm, as reported in Figs. 1 and 2. The second one, (so-called MCTS-GA), is a hybrid agent, that makes use of both MCTS and classic Genetic Algorithm, in two consecutive steps. This agent first uses MCTS to rapidly explore the solution space, starting from zero knowledge about the final solution, and looks for a sufficiently good, but still imperfect design hypothesis. As a further step, the agent makes use of a Genetic Algorithm to refine the imperfect design. In practical terms, MCTS-GA executes the pseudo-code showed in Fig. 2 as long as its best design hypothesis' fitness is lower than a certain fraction of the fitness goal (FG). After reaching this partial target (that we tuned to 80% of FG), it stops the execution of MCTS, and starts running our implementation of the pseudo-code given in Fig. 3 (i.e. GA). In doing this, the Genetic Algorithm takes as first-generation PS (i.e. population size) copies of the best design hypothesis previously elaborated by MCTS. The whole search job ends as soon as the fitness level reaches the full value of FG.

Additionally—in order to have a reference for the achieved performance levels—we included two other agents (GA1 and GA2), these being two implementations of classic GA, with different values of controlling parameters (see Sect. 4).

---

**Adopted Genetic Algorithm's pseudo-code**


---

**Input:**

- f: fitness function
- FG: fitness goal
- PS: population size
- NI: maximum number of iterations
- CR: crossover rate
- MR: mutation rate
- IR: immigration rate

**Output:**

- the best solution found, b
- number of performance evaluations, n

```

1 generate an initial population P, of size PS
2 evaluate the performance of elements of P, using f
3 while (iteration < NI && fitness < FG)
4     select the best CR·PS solutions for the crossover
5     generate PS·(1-CR-IR) new solutions with crossover
6     regroup solutions of step 4 and 5 into a new population P*
7     apply random mutations to the items of P* according to MR
8     append IR·PS randomly created solutions to P*
9     evaluate the performance of elements of P*, using f
10 end
11 find the best single solution in P*, b
12 return b, n

```

---

**Fig. 3** Genetic Algorithm's pseudo-code—adapted from [33]

### 3 Structural design tasks as search problems

When dealing with structural design tasks, engineers often have to explore a vast space of possible alternatives. But exploration comes at a cost, as developing one design instance requires (1) qualified workforce, (2) additional time to product delivery and (3) computational resources. In this sense, as long as a technical problem can be described analytically in a way the algorithm can handle, MCTS may result in being quite helpful, as it operates effectively in vast search spaces, with a limited number of instantiations [22].

Many structural engineering design problems could be treated just as search-problems with a vast solution space. We prove this, with an example. Let the work task consist of designing an earthquake-resistant  $u$ -storey reinforced concrete building, with  $v$  load-bearing elements at each storey. In this problem, the shape, size and positions of the bearing elements affect the dynamic response of the building and determine the seismic response to be considered in the calculations. We call  $w$ , the number of possible different cross-sections for a single load-bearing element at ground floor. We assume that, once the cross-section at floor  $i-1$

has been selected, the number of possible alternatives for the element immediately on top of it (i.e. that at floor  $i$ ) is immediately reduced—for consistency reasons—by a factor of  $x$ . Both variables are integers and at least equal to 1. As a consequence, for each set of  $t$  load-bearing elements on the same vertical line, the number of possible different solutions— $y$ —is as given in Formula 2.

$$y = \prod_{k=0}^{u-1} \max\left\{\left\lfloor \frac{w}{x^k} \right\rfloor, 1\right\}. \quad (2)$$

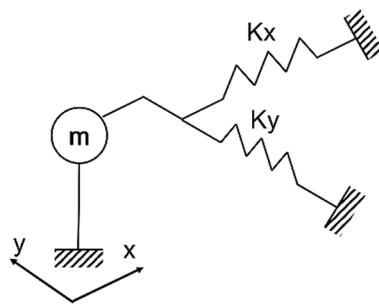
The possible different combinations of vertical elements for the whole building is equal to  $z=y^v$ . As an example, with  $u=6$ ,  $v=49$ ,  $w=36$ ,  $x=4$ , we get  $y=648$  and then  $z \approx 5.85$ , the total number of combinations would be  $10^{137}$  (for sake of brevity we ignored the design options concerning the beams). With a machine capable of evaluating  $100 \times 10^9$  different structural models each second (that is far from true, as a structural calculation can take seconds to minutes to be completed), the full exploration of this search space would take  $10^{118}$  years to be finished on a regular desktop machine.

Similar problems can be developed for steel and masonry buildings as well, this proving that search problems

concern many different applications in practical structural engineering.

### 3.1 Definition of the considered structural problem

In this paper, we consider a simple reference design problem, to show how MCTS may be integrated among structural engineering design tools. The problem consists of a one-storey RC building with a rigid horizontal slab placed on both columns and shear walls as vertical load-bearing elements. Beams are considered as part of the slab. The task is to define number, shape, size and position of the load-bearing elements, so to achieve specific targets ( $\theta$ ) in terms of the dynamic behaviour of the structure. Targets are expressed in terms of (1) a selected main translational periods ( $T_X, T_Y$ ) [45], and (2) maximum distance between the centre of mass (CM) and the centre of stiffness (CS) of the structure [45]. The problem can be simplified to a single degree of freedom (SDOF) as illustrated in Fig. 4,



**Fig. 4** SDOF system of the one-storey RC building

where (1) the mass  $m$  is obtained from both the slab's and (half of the) vertical elements' mass, while (2) stiffness values  $K_x$  and  $K_y$  depend (as an assumption) just on flexural deformability of the bearing elements. More in detail, horizontal stiffness in the direction  $i - K_i$ —is obtained as given in Formula 3 [46].

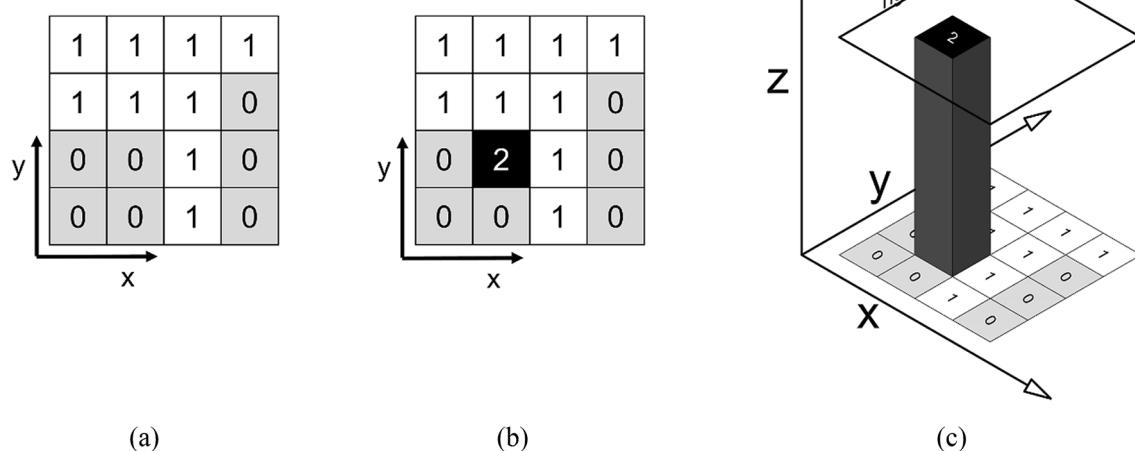
$$K_i = \sum_{s=1}^n \frac{12EJ_{s,i}}{h_s^3}, \quad (3)$$

where  $n$  is the number of RC bearing elements;  $E$  is concrete's Young Module (assumed always equal to 30,000 MPa);  $J_{s,i}$  is the moment of inertia around the axis orthogonal to direction  $i$ ;  $h_s$  is the height of the  $s$ th bearing element (always assumed equal to 3.0 m).

Once the story's lateral stiffness in the direction  $i$  is known, the correspondent vibration period— $T_i$ —can be calculated as given in Formula 4 [45].

$$T_i = 2 \cdot \pi \sqrt{\frac{m}{K_i}}. \quad (4)$$

To resemble real-life projects' requirements, while looking for suitable combinations of bearing elements, our agents have to observe rules. First of all, we assume that design units cannot be placed everywhere on the ground-floor; on the contrary—so to simulate real-life design limitations—positions exist where they are not allowed (see white-coloured 1-cells in Fig. 5); we call such limitations *geometric constraints* ( $\gamma$ ). Second, we do assume that the building's storey has a squared shape, with an assigned value of the distributed mass of  $\eta$  t/m<sup>2</sup> (these being *design*



**Fig. 5** **a** Example of a design task in a 4-by-4 grid. **b** Insertion of a bearing element in a free cell of the task's grid. **c** A 3D view of the inserted bearing element

*assumptions, v*). The set of target values  $\theta$ , geometric constraints  $\gamma$ , and design assumptions  $v$  forms a *design task*,  $\delta$ .

### 3.2 Problem's analytical description and assumptions

To be able to handle the problem analytically, we discretized the structural plan into a grid of identical squared cells; each cell hosts an integer value from the set {0, 1, 2}. Zeros (0) identify free positions, i.e. locations where a design unit can be placed; one (1) values stay for unavailable locations, i.e. cells not accessible to the agent for selection. As bearing elements are inserted, free positions are filled with the number two (2).

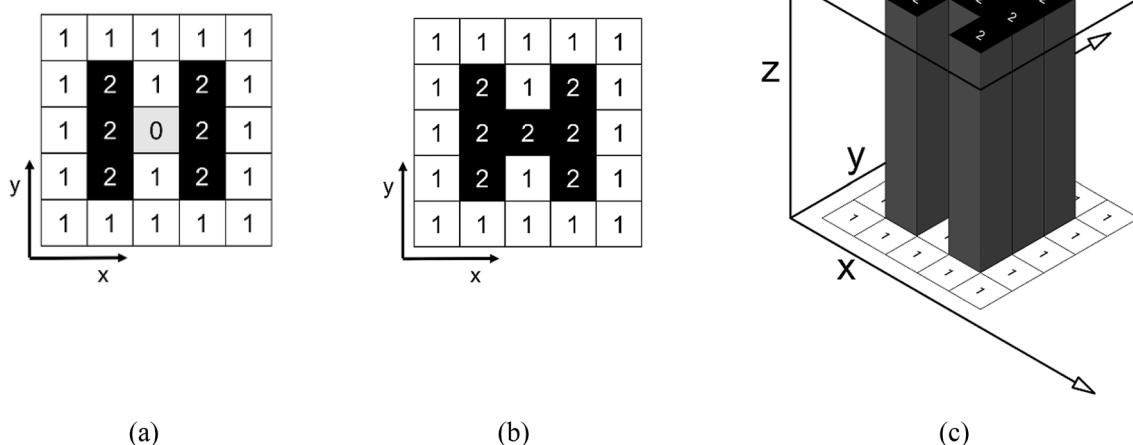
All the envisaged agents operate on the grid by changing one single cell at a time. MCTS can just add design units (in the *expansion* step)—see for example Fig. 5a–c—while GA can both add and remove them (using *crossover* and *mutation*). Every time a design unit is added or removed, the global geometry of the bearing elements changes and has to be re-computed. In addition, if any two cells come to have a side in common, they are considered as belonging to the same structural bearing element. On one hand, the insertion of a design unit in a certain position can lead to two, three or four different bearing entities (columns and/or walls) being unified into a new, larger structural element; on the other hand, the removal of a design unit may provoke disconnection and separation of one structural element in up to four new smaller and independent entities. Because of this, in our study, the agents' search space is characterised by strong discontinuity and non-linearity. As an example, Fig. 6a–c show how the number of bearing elements and as

a consequence the horizontal stiffness of the storey along the X-direction can change drastically in just one single step. In the code, a specific procedure to update the global geometry of the load-bearing elements in case of insertion or removal of a bearing units was developed and implemented. (In our study, as a work hypothesis, we set no limitations to the shape and size a bearing element can have, as long as (1) it is obtained by putting squared cells together, and (2) it is consistent with  $\gamma$ .)

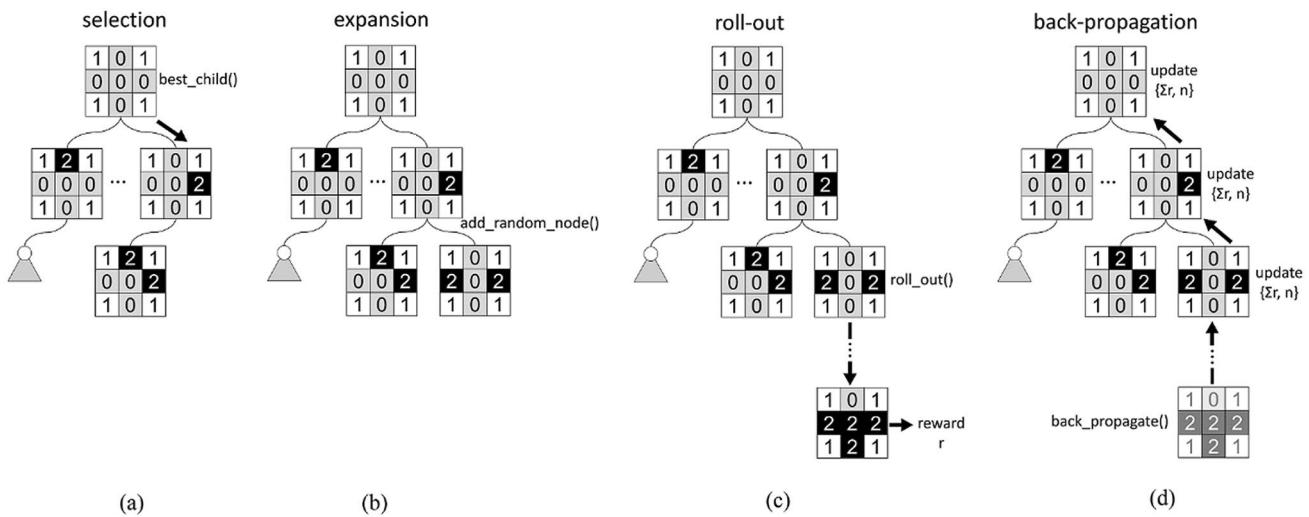
In Fig. 7, we show an example of the MCTS' four steps; it was obtained by adapting Fig. 1 to the specific search of our problem. In Fig. 7a, the tree is already partially developed; the root is not a leaf anymore, so the *best\_child()* function is called to select the most promising branch. (In the figure, a light-grey triangle with a white circle on top represents a sub-tree we omitted for the sake of clarity.) In Fig. 7b, a leaf node is expanded with the addition of a new node. In Fig. 7c, a roll-out is performed, till a reward is generated. Finally, in Fig. 7d, the reward is back-propagated, by updating the values of total reward and number of simulations performed all along the path back to the tree's root. After back-propagation (as suggested by the transparency of the node at the bottom of the tree in Fig. 7d), those nodes created within the roll-out are discarded.

### 3.3 Problem's search space

In Silver et al. [18], the MCTS application for the board game Go had to consider a search space of the order of  $10^{172}$  different configurations. In that study, the algorithm proved to work effectively, also due to its integration with Deep Neural Networks. In our work, as the first attempt at an application in the



**Fig. 6** Example of discontinuity in the design process. **a** Two distinct plain walls along the Y-direction. **b** One single H-shaped structure. **c** 3D view of the H-shaped structure



**Fig. 7** Visualisation of MCTS' four steps as envisaged in the present study

realm of structural engineering, we adopted some limitations. First of all, our search space's maximum dimension (problem's initial branching factor) is up to  $2.239^{102}$ . Such number is obtained by generating grids with a total of 340 zero-cells, and then letting the agent decide whether to place or not to place a design unit in each of them (the total dimension of the grids is always 20-by-20 cells). For each task we posed, many combinations could fit as an acceptable solution. In general, the more solutions exist for a design task, the easier for the agent to find them. Unfortunately, telling in advance how large this number is, is not trivial, also because design tasks  $\delta$  may be very different from each other. The GA-based agents will serve as a term of comparison of design task's complexity. As a second limitation, at this stage, we discarded any neural network as a support for the MCTS-agent, as well as nested algorithm's architecture existing in the current literature [34].

### 3.4 Performance measures

For all the agents, performance has to be measured each time a new structural solution is generated. On the one hand, this happens at the end of any roll-out phase. On the other hand, it is needed as new items are included in the solutions pool. Formula 5 is applied to calculate the performance of the  $i$ -th solution,  $p_i$ .

$$p_i = \max \left\{ \frac{A + B + \lambda \cdot \Gamma}{2 + \lambda}, 0 \right\}, \quad (5)$$

where

$$A = 1 - \frac{|T_{x,i} - T_x^*|}{T_x^*}, \quad (6)$$

$$B = 1 - \frac{|T_{y,i} - T_y^*|}{T_y^*}, \quad (7)$$

$T^*$  is the target translational period in X-direction;  $T_y^*$  is the target translational period in Y-direction;  $T_{x,i}$  is the actual translational period in X-direction of the  $i$ th solution;  $T_{y,i}$  is the actual translational period in Y-direction of the  $i$ th solution;  $\lambda$  is a control parameter with values in  $\{0, 1\}$ ; it is used for including or excluding the distance component from the performance assessment.  $\Gamma$  is the component explained in Formula 8;

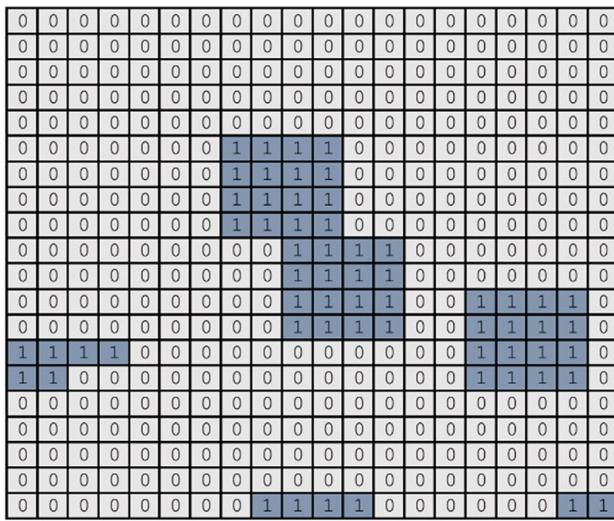
$$\Gamma = \begin{cases} 1 - \frac{d_i - d^*}{d^*}, & \text{if } d_i > d^* \\ 1, & \text{if } d_i \leq d^* \end{cases}, \quad (8)$$

$d^*$  is the target distance between the structure's centre of mass (CM) and centre of stiffness (CS);  $d_i$  is the actual distance between the structure's centre of mass (CM) and centre of stiffness (CS).

In the tests, performance goals of both 95.0% and 97.5% are used; such high level of request forces the agents to look for solutions that satisfy more than one requirement at a time.

## 4 Case studies

To test the performance of the envisaged MCTS algorithm, and to compare it with the GA-approach, four different test families are executed, each one consisting, by assumption, of two hundred different design tasks. Reporting all the inputs and outputs of the tests is not possible; in Fig. 8 we show an example of the initial arrangement of the grid in a design



**Fig. 8** Example of a design task  $\delta$  in a 20-by-20 grid

task. In the Figure, light-grey cells represent locations where bearing elements are allowed, while pale-blue cells indicate geometric constraints. Features of the design task presented in Fig. 8 are reported in Table 1 (see test family number 2), while the solutions found by the four agents (and the corresponding numerical values of the design variables) are given in Sect. 5 (both in Fig. 9 and Table 3).

The tasks' features (i.e.  $\theta$ ,  $\gamma$ , and  $v$ ) we selected for the case study were intended to make the search jobs both *meaningful* and *feasible*. First of all, the agents were expected to develop solutions by using structural elements whose dimensions are comparable to those of real buildings. Secondly, the target periods on the two directions  $X$  and  $Y$  were chosen different enough (see Table 1), so that a structural system of just squared pillars, with its symmetry of both translational mass and translational stiffness, will never represent an acceptable design solution. In other words, we made impossible for the agents to solve any design task by simply scattering a set of single design units on the grid. On the contrary, walls have to be introduced, so to differentiate the translational stiffness along with the two main directions ( $X$  and  $Y$ ). At the same time, the existence of walls requires a

progressive evolution of the design hypotheses as, in the grid we envisaged, they are quite unlikely to be obtained just by the initial random disposition of design units. (We allowed the generation of structures with RC-walls of any possible shape, i.e. also those unusual ones, commonly disregarded by engineers). Regarding feasibility, we limited the number of grid cells to 400; indeed, we adopted a grid of 20-by-20 squared cells (each one with a side of length 250 mm). Constraints were generated randomly—so to avoid biases in the design tasks—by rearranging 60 grid's cells in rectangular shapes of different proportions. As a result, in any task, there are 340 zero (light-grey) cells where the agent can place design units (in other terms, the problem's initial branching factor is 340). By taking a slab's unitary mass  $\eta$  equal to  $500 \text{ kg/m}^2$ , and setting the target vectors  $\theta$  as given in Table 1, we found four families of design problems that fit with our requirements. (In Table 1, performance goal refers to numerical values obtained by Formula 5.)

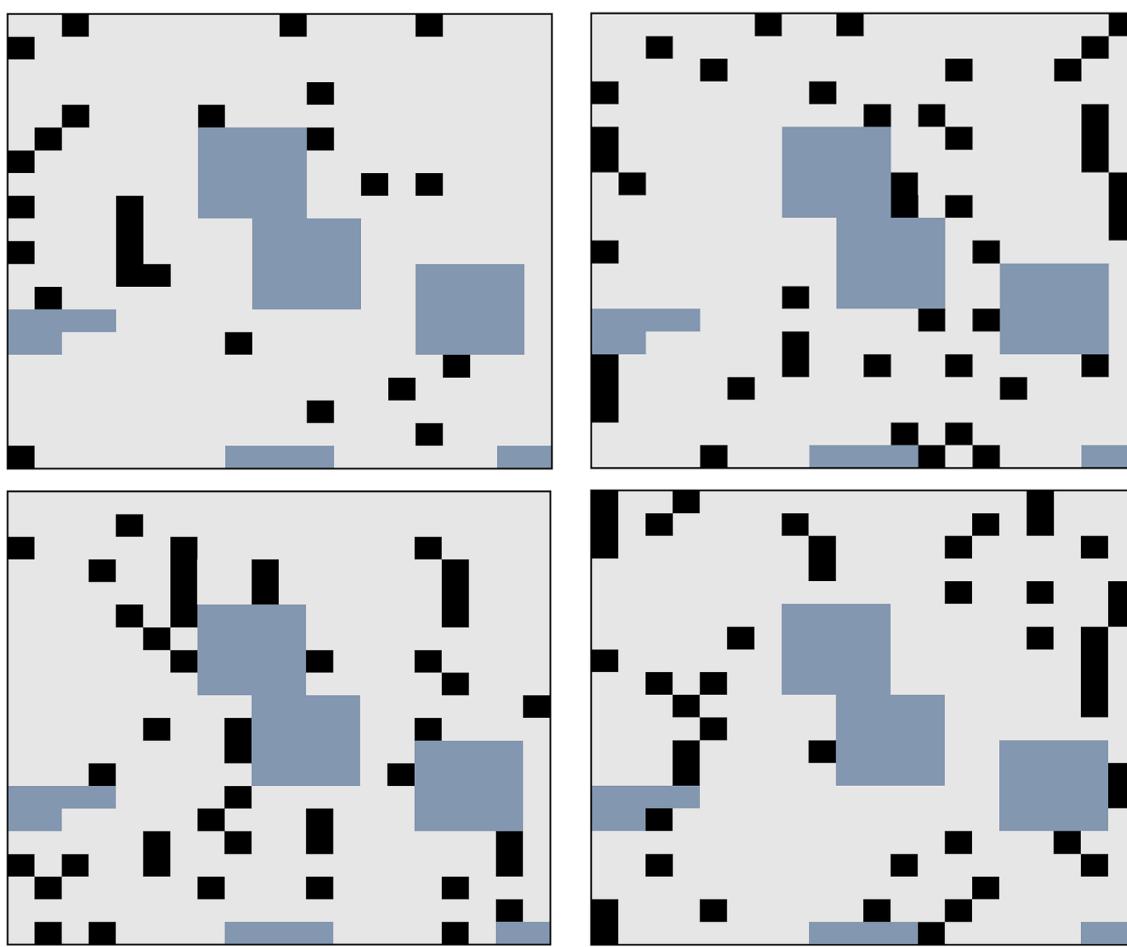
As the reader can see in Table 1, just two of the four test families include a non-zero  $\lambda$ . This means we considered the distance between the centre of mass and centre of gravity just half of the times.

For what concerns the Genetic Algorithm, as reported in Table 2, two different sets of values for the two main parameters CR and IR were adopted, so to have quite a wide spectrum for such approach. In the so-called GA1, we let the best 10% of the population survive and then generate, by crossover, 70% of the new pool of solutions. The remaining 20% is obtained by introducing brand new solutions (*immigration*). In GA2, on the contrary, 80% of a pool is obtained by immigration, while crossover represents just 10% of a pool. Both for GA1 and GA2, we set the chances of solutions developing mutations up to 20%, so to avoid the risk of search being stuck in local optima. For each solution, we set the maximum number of design units that can undergo mutation to the value of 5 (such value was tuned by observing the GA-agents' average performance on the assigned design tasks). In both cases, a population is made of 100 solutions.

For the way the tests were set, GA represents quite an effective tool for solving a given task. Indeed, solutions are obtained by progressively adjusting cells on the board. As GA has the faculty of adding and removing design units

**Table 1** List of design tasks' features considered for the case studies

Test family	Problem's dimension	$\eta$	$\theta$			$\lambda$	fitness goal
			$T_x^*$	$T_y^*$	$d^*$		
#	(–)	(t/m <sup>2</sup> )	(s)	(s)	(cell)	(–)	(–)
1	340	0.50	0.0750	0.0375	–	0	0.950
2	340	0.50	0.0750	0.0375	4	1	0.950
3	340	0.50	0.0750	0.0375	–	0	0.975
4	340	0.50	0.0750	0.0375	4	1	0.975



**Fig. 9** Example of the four agents' solutions for a design task of test family #2. **a** MCTS'; **b** GA1's; **c** GA2's; **d** MCTS-GA's

**Table 2** Parameters' values for the two GA-agents

Name (-)	CR (%)	IR (%)
GA1	10	20
GA2	10	80

(by using crossover and mutation), it can explore the search space without any need to look for moves in the future. On the contrary, MCTS can just *add* design units to the board

and, if one of them has to be excluded from the final solution, the agent has to develop a new sub-branch of the search tree (that requiring more roll-outs).

#### 4.1 Comparing results

The discussed methodology could be adopted for two main search targets in practical engineering applications. On the one hand, the discovery of convenient, otherwise inaccessible technical solutions; on the other hand, keeping

**Table 3** Parameters and number of design hypotheses ( $H$ ) of the solutions found by the four agents for the example design task

Agent (-)	Mass (t)	$K_X$ (kN/m)	$K_Y$ (kN/m)	$T_{X,i}$ (s)	$T_{Y,i}$ (s)	$d_i$ (m)	$p_i$ (-)	$H$ (-)	No. of design units placed (-)
MCTS	18.59	154,514	467,014	0.0689	0.0396	0.95	0.954	5083	26
GA1	23.05	195,313	583,938	0.0683	0.0394	0.25	0.953	1315	45
GA2	23.05	195,313	690,104	0.0683	0.0363	0.11	0.959	1617	44
MCTS-GA	23.28	199,653	720,486	0.0678	0.0357	0.50	0.952	653	46

computation-time to solution in line with conventional approaches. In this work, a first research step on the topic, we considered a relatively simple design task. For this reason, in the following we take just computation time as a key factor to show the methodology's potential. In this regard, wanting to confront the results of MCTS and GA, we had to find a meaningful shared metric, as the two algorithms work on different data structures. On one hand, MCTS creates *trees*; on the other hand, GA generates *pools* of parallel solutions belonging to the same generation. In our study, each *design hypothesis* undergoes just one single calculation. (In MCTS, this happens every time a roll-out is executed; in GA, we have this at each fitness-evaluation step. In order not to overestimate the calculations needed for the GA-approach, we stopped evaluating a generation, as soon as one of its solutions overcomes the performance goal.)

In real life engineering, a single structure may require tens of analyses—whose duration is in the order of minutes—to be fully tested against code requirements; this is the case, for example, of modern pushover procedures [45] when conducted in compliance with current building codes' standards (e.g. [47]). Computation time could be considered as a linear function of the number of hypotheses actually tested. In addition, when compared to structural calculations, auxiliary software modules concerning tree descent, back-propagation of performance, generation of new pools of solutions, or mutation, are negligible in terms of computational effort. For this reason, when comparing MCTS and GA, we took as a metric just the number of design hypotheses tested before a suitable solution is found, in the following referred to as  $H$ . In general, the larger the value of  $H$ , the more numerous the design hypotheses an agent needed to reach the performance goal in a design task. In other words,  $H$  represents the inverse of the effectiveness of an agent on an assigned search job.

In the tests, we did not provide our search agents with any initial knowledge about where to look for solutions; nonetheless, to limit the number of design hypotheses as much as possible and make the two agents based on MCTS actually competitive with GA, we had to add two basic heuristics: first, the maximum number of moves that can be taken from the root to a terminal state was assumed equal to 50. Second, the maximum number of child-nodes (at any step of the tree descent) was set equal to 4—this determines at which point a node stops being expandable (see Sect. 2 and Fig. 2). The first limit is obtained considering that no design task we posed requires more than 50 design units on the grid to be solved. The second restriction, whose numerical value was fine-tuned by trials, is intended to keep the search tree reasonably narrow, especially at first steps of tree expansion, i.e. at a stage when almost every choice can be considered acceptable by the agent, as there is limited or no feedback about the quality of the moves. Putting no limitations in the

initial expansion would lead to a ramification of the search tree up to the branching factor, so compromising the agent's performance in terms of tested design hypotheses. Also, we considered that, for the way the problem is posed, the performance of a whole design hypothesis is only limitedly dependent on the very first moves of the design agent.

The proposed limits do not represent a reasonable choice for every possible implementation of MCTS in structural engineering. On the contrary, for different design problems and with different performance metrics, they should be reconsidered.

Tests' results are presented and discussed in the following section.

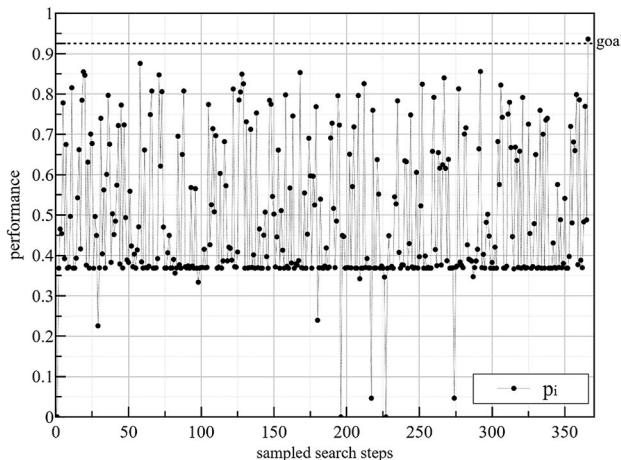
## 5 Results

In the analyses, we grouped tasks by families; each family consists of 200 comparable design tasks, i.e. problems we produced by using the same values for both *design assumptions* ( $v$ ) and *target values* ( $\theta$ ) (see Table 1), and by generating different combinations of same-dimension *geometric constraints* ( $\gamma$ ). Each algorithm ran once per each task, i.e. 200 times per task family. Neither time limitations nor memory budget were set on the analyses.

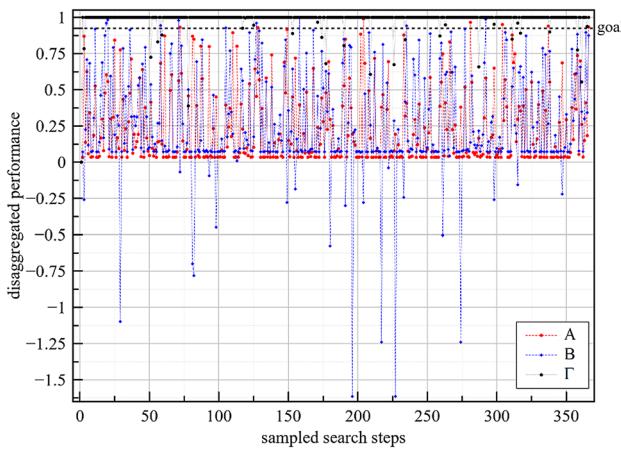
To provide the reader with reference solutions, to be used in future studies, in Fig. 9 we report the four agents' solutions for the design task of test family #2 already presented in Sect. 4; in light grey the area where bearing elements could be placed; in pale blue the geometric constraints; in black the bearing elements placed by the agent (i.e. the structural solution found). In Table 3 we show the parameters of the solutions found by the four agents in the example design task.

Looking at Table 3, it is clearly visible how MCTS-GA outperformed the other three agents in terms of number of design hypotheses ( $H$ ) needed to reach target performance values ( $p_i$ ). On the contrary, the MCTS-agent has the worst performance in this regard; nonetheless, the structural solution found by MCTS is more refined, as it is obtained with the smallest number of design units on the grid (also thanks to the presence of an “L-shaped” wall). This second design aspect is more related to the overall quality of the design. However, in the following, we will evaluate results just in terms of  $H$ , as comparing quality of solutions would require a much larger investigation, well beyond the scope of this first research step.

As a further step, we report an example of the performance pattern we obtained from the MCTS-agent: on the one hand, Fig. 10 shows numerical values obtained as per Formula 5 (i.e. agent's overall performance). On the other hand, Fig. 11 presents values we obtained for overall



**Fig. 10** Example of sampled path for overall performance of the MCTS-agent



**Fig. 11** Example of a sampled path of performance components A, B and  $\Gamma$  for the MCTS-agent (as per Formulas 6, 7 and 8)

performance's components (Formulas 6, 7 and 8). On both cases, values were sampled every 10 design hypotheses.

From the figures, we notice two things. First, the values of performance oscillate quite till the end of the analyses. This depends (1) on the non-linearity of the problem and on the discontinuity of the search space (the addition on one design unit can dramatically change the performance), (2) on the random nature inherent in the methodology (that influences the *rollout* phase), and also (4) on the need for balancing *exploration* and *exploitation* (see the two terms included within the curly brackets of Formula 1) within the search. Because of the latter, as MCTS algorithm performs the search, it outputs design hypotheses that can have very different performance values, despite being released one after the other. Interestingly, in Fig. 10 the value of performance level is quite high since the very first steps of the search (i.e. it is greater than 80% after 10

samples, that means 100 design hypotheses). The choice we made of adopting a hybrid agent, that uses MCTS on the first part of a search job (rapidly exploring the search space), and then finalizes the design using GA is due to this experimental evidence we obtained.

Second, the numerical values the agent obtained from Formulas 6, 7 and 8 have different ranges. Among the sampled points,  $\Gamma$  is on average quite close to the value of 1.0; this depends both on the target distance range  $d^*$  we selected for the example (i.e. 1.0 m) and on the overall symmetric disposal of bearing elements, randomly added by the agent during the process. Some values of variable B are negative; this does not happen with values of A. The reason is that being the period target value in direction Y half than that in direction X ( $T_y^* = 0.0375$  s,  $T_x^* = 0.0750$  s), B's positive codomain is half of A's. As a consequence, the random process governing the agent's search is more likely to produce negative values for variable B.

For what concerns overall tests' results, in Table 4 we show how MCTS-GA outperformed all the other three approaches, in terms of a smaller amount of design hypotheses needed to solve an assigned task ( $H$ ). (GA1 can be considered the best among the other three agents.)

In Table 5 we show essential statistics about variable  $H$ , for the four agents. While reporting results, we included the estimation of both mean ( $m_H$ ) and standard deviation ( $s_H$ ) of the variable; this makes possible for the reader to understand overall agents' results on the four test families of 200 search jobs each. In addition, to extend the validity of results beyond the limits of the hundreds of design tasks here considered, in Table 5 we included lower and upper bounds ( $\xi_{inf}$  and  $\xi_{sup}$ ) of the 95%-confidence interval of  $H$  (computed as per Formula 9). Such bounds make it possible to obtain intervals where real mean values of variable  $H$  (i.e.  $\mu$ ) have 95% probability to be (intervals are graphically reported in Fig. 12).

$$\xi_{inf} = m - 1.96 \frac{s}{\sqrt{n}} \leq \mu \leq m + 1.96 \frac{s}{\sqrt{n}} = \xi_{sup}, \quad (9)$$

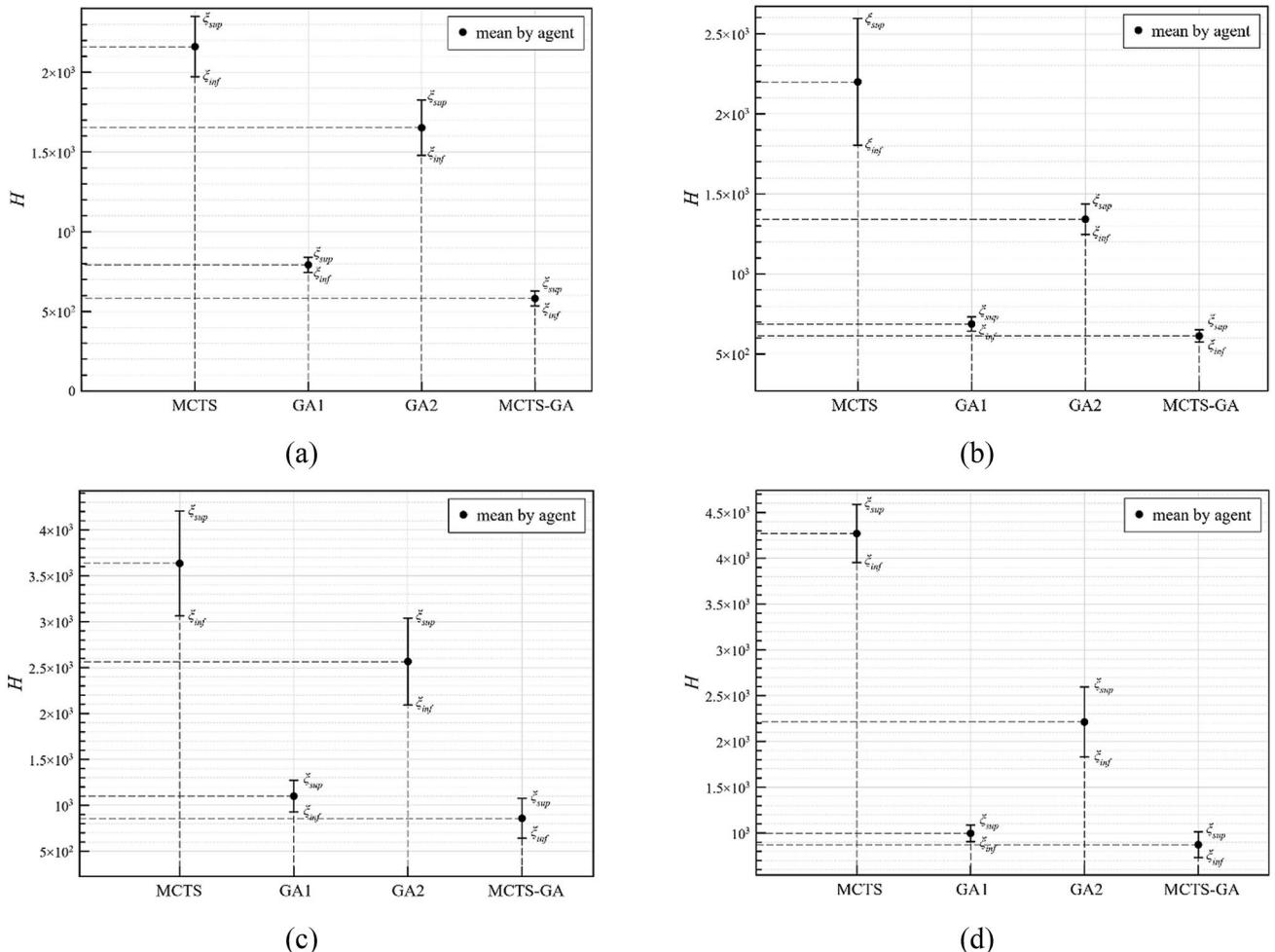
where  $m$  is the average number of design hypotheses needed to reach the performance goal;  $s$  is the sample's standard

**Table 4** Relative number of times MCTS-GA outperformed MCTS, GA1 and GA2, respectively

Tasks family	MCTS-GA's wins over		
	MCTS (%)	GA1 (%)	GA2 (%)
1	90.0	72.0	91.5
2	79.0	58.0	83.5
3	97.0	67.0	91.5
4	97.5	58.5	95.0

**Table 5** Mean and standard deviation values of the number of design hypotheses needed ( $H$ ) by the four agents

Test family	MCTS				GA1				GA2				MCTS-GA				
	#	$m_H$	$s_H$	$\xi_{inf}$	$\xi_{sup}$												
1		2161	1365	1972	2350	792	342	744	839	1652	1246	1479	1825	581	338	535	628
2		2199	2855	1803	2594	668	317	644	732	1342	669	1245	1438	613	272	575	650
3		3634	4120	3063	4205	1100	1237	928	1271	2566	3399	2095	3037	858	1561	641	1074
4		4270	2286	3954	4587	997	664	905	1089	2212	2754	1830	2593	873	1009	733	1012

**Fig. 12** Average values and 95%-confidence intervals of the number of design hypotheses tested before a suitable solution is found ( $H$ ) for a given design task ( $\delta$ ); **a** test family #1; **b** test family #2; **c** test family #3; **d** test family #4

deviation, of variable  $H$  (i.e. of the number of design hypotheses needed to reach the performance goal); this is taken as an estimator of the actual standard deviation of the corresponding distribution,  $\sigma$ ;  $n$  is the number of times the algorithm has run;  $\mu$  is the average value of the real distribution, within the 95% confidence interval.

From data in Table 5, it clearly emerges how, for the proposed set of design problems, MCTS-GA always presents the lowest values for the mean of  $H$ . On the contrary, its

standard deviation is not always the smallest (as GA1 has a smaller one in test families 3 and 4). Looking at the results data, we noticed this is due just to a limited number of tasks where MCTS-GA needed a relatively large number of design hypotheses.

Values of Table are graphically depicted in Fig. 12. From the figure, the reader can understand, at a glance, how MCTS-GA represents the best option for solving a task of any of the four families.

Data we reported in Fig. 12 could be disaggregated into single data points; an example, regarding test family 1, is given in Fig. 13. Unfortunately, from such kind of representation is not clearly emerging how well one agent is performing in comparison to others. So, to have a clearer picture of the whole performance of the four agents, in Fig. 14 we plotted again their values of  $H$ ; this time, we sorted each set of data by value. In this way, correspondence between ordinates at the same abscissa is lost, but we get a more understandable information about how the MCTS-GA-agent resulted more efficient in producing results for each batch of tasks as a whole. Indeed, for the vast majority of the tasks, its ordinates clearly stay below those of the other three agents.

Results we included on both tables and charts of this section clearly show that a plain, 1-player version of MCTS, for what concerns computation time, is far from being competitive in the problem we posed. On the contrary, an agent based on Genetic Algorithm, whose values of IR and CR are those of GA1 (see Table 1), performs very well. Nonetheless, a hybrid agent, obtained from both MCTS and GA, outperformed all the other approaches we considered.

## 6 Limitations

The here-discussed work is not free from limitations.

First of all, we assumed that it is possible to describe a real-life structural design problem by using a grid of squared cells, so to resemble a representation often used in board- and video-games. More complex engineering tasks (like designing a multi-storey building), may require different approaches (e.g. also considering imperfect knowledge)

and more extensive simulations (e.g. a stack of grids to be operated in parallel). Despite the software we developed is far from being a ready-to-use tool, its algorithmic kernel will stay the same in future applications.

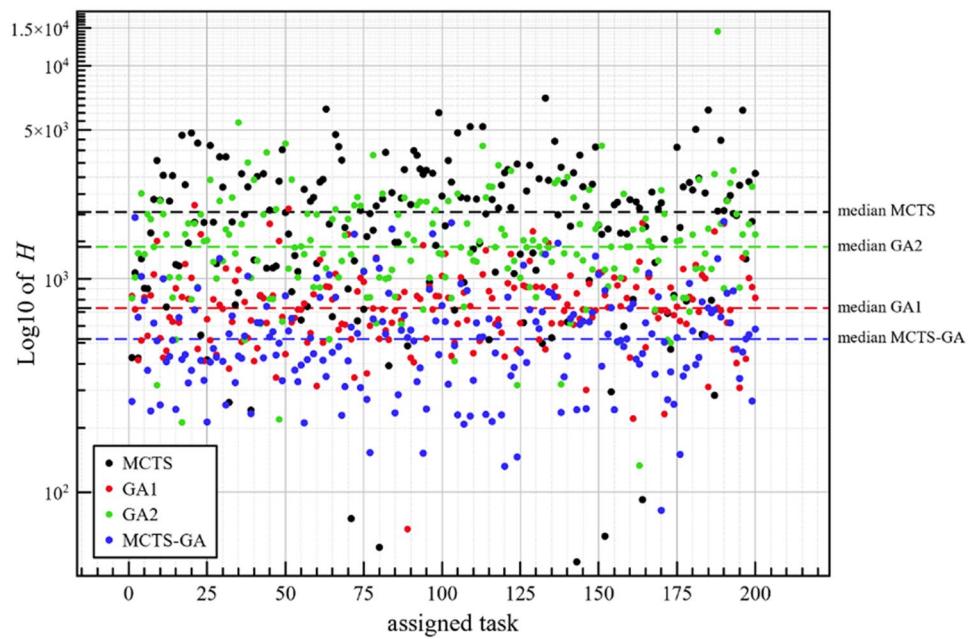
Second, we took as assumptions, that (1) design performance is deterministically obtainable once the structural model is given, and that (2) material properties are uniform everywhere. Introducing uncertainty and non-uniformity will enrich the search space dramatically.

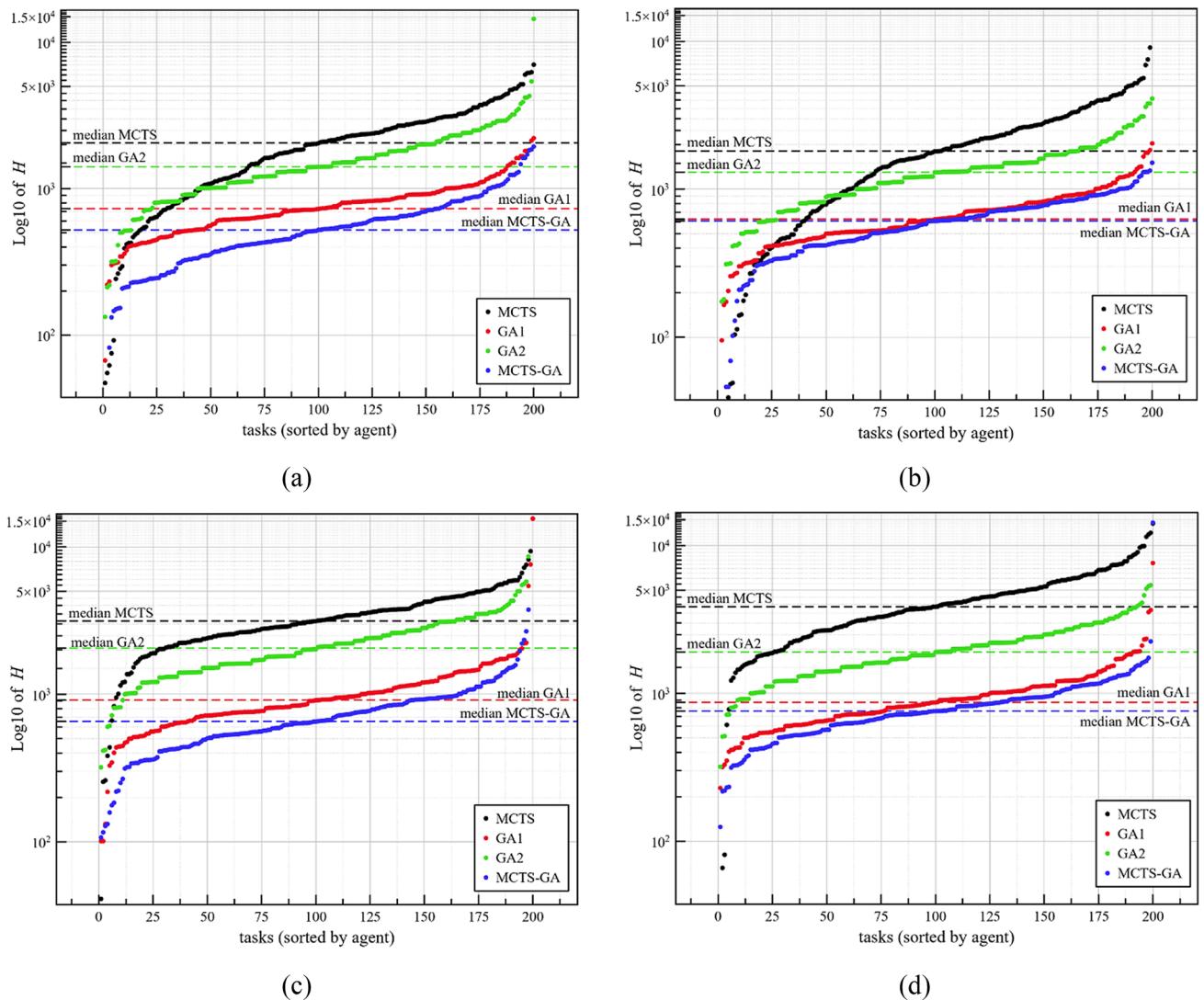
Third, as an initial research step, we kept the maximum number of possible alternatives in the order of  $2.239^{10^2}$ . With the help of high-capability computational resources, such value may be enlarged in future studies.

Fourth, we computed horizontal stiffness as a function of the sole flexural deformability and we considered all the load bearing elements as liner elastic. This may affect the straightforward application of the developed code in practical design problems but is irrelevant for what we wanted to prove in terms of engineering problems' searchability with MCTS. Future developments may reduce the distance to real-life implementations, by taking into consideration, for example, buildings' non-linear structural response and/or performance-based design. Such further step will require, under the same algorithmic approach, a revision of both the here-discussed performance goals (e.g., by introducing terms as *inelastic structural response* or *expected seismic consequences*) and the correspondent performance measures (e.g. *inelastic vibration periods*, and *expected annual losses* respectively).

Fifth, real applications in structural engineering are more complex and have to consider specific architectural demands (e.g. position of the elevator shaft) and constructive rules

**Fig. 13** Values of  $H$  for the four agents—example taken from test family #1





**Fig. 14** Number of design hypotheses needed to solve the task— $H$ —sorted for each agent. **a** Test#1. **b** Test #2. **c** Test #3. **d** Test #4

(e.g. maximum spacing of load-bearing elements) which has to be translated into constraints. However, the principles to consider these restraints were illustrated and it can be expected that such constraints will reduce the number solutions and as a consequence the computational time.

Sixth, as a metric to evaluate solutions' quality, we took into consideration the variable  $H$  (i.e. the number of design hypotheses an agent needed before reaching the performance target); at the same time, we ignored the complexity of the structural solutions proposed by the agents as a possible term of comparison. This happened as, for this first research step, we considered computation time a clearer metric than overall design quality.

Finally, in the study we considered (1) a plain version of MCTS, (2) a classic approach to the Genetic Algorithm, and (3) a combination of the two. This means that study's

results cannot be generalized to all the possible variants of the presented methodologies.

## 7 Conclusions

In civil structural engineering, design tasks exist that can be reduced to search problems. In this paper, we showed how the task of generating structural solutions with specific dynamic properties can be faced by adopting both MCTS and GA. On the one hand, the presented approach is effective, as it found acceptable solutions for the envisaged technical problems. On the other hand, it is efficient, as it outperformed the prominent GA technique in terms of necessary design hypotheses.

As a first step, we introduced the MCTS algorithm; then, we presented the pseudo-codes we implemented from scratch in language code Java. Secondly, we briefly showed how structural design problems can easily have an extremely vast search space. Third, we described a suitable problem for a single-storey RC building, which resembles design tasks engineers face in their work. The problem is characterised by both non-linearity and discontinuity. Then, we focused on a set of case studies, for each of which we run hundreds of instantiations using both Monte Carlo Tree Search and Genetic Algorithm. On the one hand, the pattern the plain MCTS-agent followed to reach solutions for the selected problems looked far from linear and continuous, also because the problem's nature. On the other hand, such agent is capable of reaching a quite high value of performance very soon on the analyses. As a consequence, a hybrid agent, composed of an initial involvement of MCTS and later intervention of GA seems to be a successful strategy. Indeed, the results we obtained show that, with regard to computation time, plain MCTS is, by far, not the best option for the posed structural problem; nonetheless, a hybrid agent that implements MCTS and GA proved to be the most efficient tool. In our study, on average, it needed to take into consideration from 1.09 to 2.99 times less design hypotheses compared to GA, so to reach a performance point beyond the goal we set. Interestingly, this may result in extremely important in those cases—as real-life structural engineering—where simulation is time-consuming.

Our study is far from offering an exhaustive comparison between MCTS and GA. Nonetheless, it proved how, in some cases a hybrid implementation of both can provide efficient solutions to a structural engineering problem. In conclusion, this work has revealed that MCTS may become a very useful design tool for civil engineers, especially as a member of a toolbox of existing Artificial Intelligence techniques.

Future research will be aimed at reducing the limitations we discussed in the text (also implementing enhancements [41] beyond the plain, non-adversarial version of MCTS), and at expanding our reach in terms of engineering applications (i.e. including more complex and more realistic structures). In addition, other metrics rather than just number of tested design hypotheses (as for example *design quality*) may be introduced to expand the reach of this research work. Finally, other Monte Carlo search algorithms, as Nested Roll-out Policy Adaption (nRPA) [48] and Nested Monte Carlo Search (NMCS) [49] may represent a key to achieve further advancements in our research.

**Electronic supplementary material** The online version of this article (<https://doi.org/10.1007/s00366-021-01338-2>) contains supplementary material, which is available to authorized users.

**Funding** Open access funding provided by Università degli Studi di Perugia within the CRUI-CARE Agreement.

**Open Access** This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by/4.0/>.

## References

1. Norvig P, Russel S (2009) Artificial Intelligence: a Modern Approach. 3rd Edition, Pearson, ISBN-13 978-9332543515
2. Sriram D (1987) Knowledge-based Approaches for Structural Design. Computational Mechanics Publications, UK, 164 pp, ISBN 0 905451 78 3
3. Maher ML, Fenves SJ, Garrett JH (1988) Expert systems for structural design. Expert Systems in Construction and Structural Engineering, Chapman and Hall
4. Arciszewski T (1992) The inductive system: a new tool in civil engineering. In: Topping BHV (ed) Optimization and artificial intelligence in civil and structural engineering. NATO ASI Series (Series E Applied Sciences), vol 221. Springer, Dordrecht. [https://doi.org/10.1007/978-94-017-2492-0\\_17](https://doi.org/10.1007/978-94-017-2492-0_17)
5. Arciszewski T, Ziarko W (1990) Inductive learning in civil engineering: a rough sets approach. Microcomput Civil Eng 5(1):19–28
6. Torky AA, Aburawwash AA (2018) A deep learning approach to automated structural engineering of prestressed members. Int J Struct Civ Eng Res 7(4):347–352
7. Dias WPS, Weerasinghe RLD (1996) Artificial Neural Networks for construction bid decisions. Civ Eng Syst 13(3):239–253. <https://doi.org/10.1080/02630259608970200>
8. Cascardi A, Micelli F, Aiello M (2016) Analytical model based on Artificial Neural Network for masonry shear walls strengthened with FRM systems. Compos B Eng 95:252–263. <https://doi.org/10.1016/j.compositesb.2016.03.066>
9. Fahmy AS, El-Madawy ME, Gobran YA (2016) Using artificial neural networks in the design of orthotropic bridge decks. Alexandria Eng J 55(4):3195–3203 (Elsevier)
10. Lorterapong P, Rattanadamrongagsorn T (2001) Viewing construction scheduling as a constraint satisfaction problem. In: Topping BHV, Kumar B (eds) Proceedings of the Sixth International Conference on the Application of Artificial Intelligence to Civil and Structural Engineering. Civil-Comp Press, Stirlingshire, UK, Paper 8, 2001. <https://doi.org/10.4203/ccp.74.8>
11. Chisari C, Amadio C (2018) TOSCA: a tool for optimisation in structural and civil engineering analyses. Int J Adv Struct Eng 10:401–419. <https://doi.org/10.1007/s40091-018-0205-1>
12. Amadio C, Fragiacomo M, Lucia P, Luca OD (2008) Optimized design of a steel-glass parabolic vault using evolutionary multi-objective algorithms. Int J Space Struct 23(1):21–33
13. Islam MS, Rokonuzzaman M (2018) Optimized design of foundations: an application of genetic algorithms. Aust J Civ Eng 16:2018–2021

14. Hudson MG, Parmee IC (1995) The application of genetic algorithms to conceptual design. In: Sharpe J (ed) AI system support for conceptual design. Springer-Verlag, Berlin, pp 17–36
15. Rafiq Y, Sui C, Zhou GC, Easterbrook D, Bugmann G (2005) Using artificial intelligence techniques to predict the behaviour of masonry panels. <https://doi.org/10.4203/ccp.82.21>
16. Zhou GC, Rafiq Y, Bugmann G, Easterbrook D (2006) Cellular automata model for predicting the failure pattern of laterally loaded masonry wall panels. *J Comput Civ Eng.* [https://doi.org/10.1061/\(ASCE\)0887-3801\(2006\)20:6\(400\)](https://doi.org/10.1061/(ASCE)0887-3801(2006)20:6(400))
17. Krish S (2011) A practical generative design method. *Comput-Aid Design* 43(1):88–100 (**Elsevier**)
18. Silver D et al (2016) Mastering the game of Go with deep neural networks and tree search. *Nature* 529:484–489
19. Winands MHM (2015) Monte-Carlo Tree Search. In: Lee N (ed) Encyclopedia of computer graphics and games. Springer, Cham, pp 1–6
20. Aggarwal CC (2018) Neural Networks and deep learning. Springer, Berlin, p 497 (**ISBN 978-3-319-94462-3**)
21. Chikun C (2010) Go: A complete introduction to the game. Kiseido Publishing Co; 1st edn.
22. Silver D, Hubert T, Schrittwieser J, Antonoglou I et al (2018) A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. *Paper Supplementary Mater Sci* 362(6419):1140–1144. <https://doi.org/10.1126/science.aar6404>
23. Broderick A, Ryan BH, Philip H (2010) Monte Carlo Tree Search in Hex. *IEEE Trans Comput Intell AI Games* 2(4):251–258
24. Mehat J, Cazenave T (2011) A parallel general game player. *KI J* 25(1):43–47
25. Winands MHM (2017) MCTS in board games. *Handbook of digital games and entertainment technologies*. Springer, Singapore, pp 47–76
26. Evans R, Jumper J, Kirkpatrick J, Sifre L, Green T, et al. (2018) De novo structure prediction with deep-learning based scoring. In Thirteenth Critical Assessment of Techniques for Protein Structure Prediction (Abstracts) 1–4 December 2018
27. Segler MHS, Preuss M, Waller MP (2018) Learning to plan chemical syntheses. *Nature* 555:604–610
28. Trunda O, Barták R (2013) Using Monte Carlo Tree Search to solve planning problems in transportation domains In: *Advances in soft computing and its applications*. Springer, Berlin, pp 435–449
29. Edelkamp S, Gath M, Greulich C, Humann M, Herzog O, Lawo M (2016) Monte-Carlo Tree Search for logistics. In: *Commercial transport*. Springer International Publishing, Berlin, pp 427–440
30. Sironi CF (2019) Monte-Carlo Tree Search for artificial general intelligence in games, Proefschriftmaken.nl || Uitgeverij BOX-Press. <https://doi.org/10.26481/dis.20191113cs>
31. Dieb S, Ju S, Shiomi J, Tsuda K (2019) Monte Carlo tree search for materials design and discovery. *MRS Commun.* <https://doi.org/10.1557/mrc.2019.40>
32. Gaymann A, Montomoli F (2019) Deep Neural Network and Monte Carlo Tree Search applied to fluid-structure topology optimization. *Sci Rep.* <https://doi.org/10.1038/s41598-019-51111-1>
33. Mitchell M (1996) An introduction to genetic algorithms, pp. 205, Bradford Books, ISBN-10 0262133164
34. Browne C et al (2012) A survey of MCTS methods. *IEEE Trans Comput Intell AI Games* 4(1):1–43
35. Chaslot GMJ-B et al (2008) Progressive strategies for Monte-Carlo Tree Search. *New Math Nat Comput* 4(3):343–359
36. Sironi CF, Winands MHM (2016) Comparison of rapid action value estimation variants for general game playing. In: 2016 IEEE Conference on Computational Intelligence and Games (CIG 2016), pp 309–316
37. Arneson B, Hayward RB, Henderson P (2010) Monte Carlo Tree Search in Hex. *IEEE Trans Comput Intell AI Games* 2(4):251–258
38. Pepels T, Winands MHM, Lanctot M (2014) Real-time Monte Carlo Tree Search in Ms Pac-Man. *IEEE Trans Comput Intell AI Games* 6(3):245–257
39. Silver D, Schrittwieser J, Simonyan K et al (2017) Mastering the game of Go without human knowledge. *Nature* 550(7676):354
40. Kocsis L, Szepesvári C (2006) Bandit based Monte-Carlo planning. In: Fürnkranz J, Scheffer T, Spiliopoulou M (eds) *Machine learning: ECML 2006, 17th European Conference on Machine Learning, Berlin, Germany, September 18–22, 2006, proceedings. Lecture notes in computer science*, vol 4212. Springer, Berlin, pp 282–293. [https://doi.org/10.1007/11871842\\_29](https://doi.org/10.1007/11871842_29) (**ISBN 3-540-45375-X**)
41. Schadd MPD, Winands MHM, Tak MJW, Uiterwijk JWHM (2012) Single-Player Monte-Carlo Tree Search for same game. *Knowl-Based Syst* 34:3–11
42. Holland J (1992) Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence, Bradford Books, 1992 ISBN 0262581116
43. Koza JR (1992) *Genetic programming: on the programming of computers by means of natural selection*. MIT Press, Cambridge
44. Renner G, Ekárt A (2003) Genetic algorithms in computer aided design. *Comput Aided Des* 35(8):709–726
45. Chopra AK (2016) *Dynamics of Structures: Theory and Applications to Earthquake Engineering*, pp. 960, Pearson College Division, ISBN-10 9780134555126, 2016
46. Wilson EL (1998) Three dimensional static and dynamic analysis of structures: a physical approach with emphasis on earthquake engineering, Ed. CSi.
47. MIT – Ministero delle Infrastrutture e dei Trasporti (2019) Decreto Ministeriale del 14 gennaio 2008 - Aggiornamento delle «Norme tecniche per le costruzioni». GU Serie Generale n.42 del 20-02-2018 - Suppl. Ordinario n. 8 (**in Italian**)
48. Rosin CD (2011) Nested rollout policy adaptation for Monte Carlo Tree Search. *IJCAI 2011:649–654*
49. Cazenave T (2009) Nested Monte-Carlo Search. *IJCAI 2009:456–461*

**Publisher's Note** Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

## Terms and Conditions

Springer Nature journal content, brought to you courtesy of Springer Nature Customer Service Center GmbH (“Springer Nature”). Springer Nature supports a reasonable amount of sharing of research papers by authors, subscribers and authorised users (“Users”), for small-scale personal, non-commercial use provided that all copyright, trade and service marks and other proprietary notices are maintained. By accessing, sharing, receiving or otherwise using the Springer Nature journal content you agree to these terms of use (“Terms”). For these purposes, Springer Nature considers academic use (by researchers and students) to be non-commercial.

These Terms are supplementary and will apply in addition to any applicable website terms and conditions, a relevant site licence or a personal subscription. These Terms will prevail over any conflict or ambiguity with regards to the relevant terms, a site licence or a personal subscription (to the extent of the conflict or ambiguity only). For Creative Commons-licensed articles, the terms of the Creative Commons license used will apply.

We collect and use personal data to provide access to the Springer Nature journal content. We may also use these personal data internally within ResearchGate and Springer Nature and as agreed share it, in an anonymised way, for purposes of tracking, analysis and reporting. We will not otherwise disclose your personal data outside the ResearchGate or the Springer Nature group of companies unless we have your permission as detailed in the Privacy Policy.

While Users may use the Springer Nature journal content for small scale, personal non-commercial use, it is important to note that Users may not:

1. use such content for the purpose of providing other users with access on a regular or large scale basis or as a means to circumvent access control;
2. use such content where to do so would be considered a criminal or statutory offence in any jurisdiction, or gives rise to civil liability, or is otherwise unlawful;
3. falsely or misleadingly imply or suggest endorsement, approval , sponsorship, or association unless explicitly agreed to by Springer Nature in writing;
4. use bots or other automated methods to access the content or redirect messages
5. override any security feature or exclusionary protocol; or
6. share the content in order to create substitute for Springer Nature products or services or a systematic database of Springer Nature journal content.

In line with the restriction against commercial use, Springer Nature does not permit the creation of a product or service that creates revenue, royalties, rent or income from our content or its inclusion as part of a paid for service or for other commercial gain. Springer Nature journal content cannot be used for inter-library loans and librarians may not upload Springer Nature journal content on a large scale into their, or any other, institutional repository.

These terms of use are reviewed regularly and may be amended at any time. Springer Nature is not obligated to publish any information or content on this website and may remove it or features or functionality at our sole discretion, at any time with or without notice. Springer Nature may revoke this licence to you at any time and remove access to any copies of the Springer Nature journal content which have been saved.

To the fullest extent permitted by law, Springer Nature makes no warranties, representations or guarantees to Users, either express or implied with respect to the Springer nature journal content and all parties disclaim and waive any implied warranties or warranties imposed by law, including merchantability or fitness for any particular purpose.

Please note that these rights do not automatically extend to content, data or other material published by Springer Nature that may be licensed from third parties.

If you would like to use or distribute our Springer Nature journal content to a wider audience or on a regular basis or in any other manner not expressly permitted by these Terms, please contact Springer Nature at

[onlineservice@springernature.com](mailto:onlineservice@springernature.com)