



HAL
open science

Combining Monte Carlo Tree Search and Depth First Search Methods for a Car Manufacturing Workshop Scheduling Problem

Valentin Antuori, Emmanuel Hébrard, Marie-José Huguet, Siham Essodaigui,
Alain Nguyen

► **To cite this version:**

Valentin Antuori, Emmanuel Hébrard, Marie-José Huguet, Siham Essodaigui, Alain Nguyen. Combining Monte Carlo Tree Search and Depth First Search Methods for a Car Manufacturing Workshop Scheduling Problem. International Conference on Principles and Practice of Constraint Programming, Oct 2021, Montpellier (on line), France. 10.4230/LIPICs.CP.2021.14 . hal-03372005

HAL Id: hal-03372005

<https://laas.hal.science/hal-03372005>

Submitted on 9 Oct 2021

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Combining Monte Carlo Tree Search and Depth First Search Methods for a Car Manufacturing Workshop Scheduling Problem

Valentin Antuori ✉

Renault, LAAS-CNRS, Université de Toulouse, CNRS, Toulouse, France

Emmanuel Hebrard ✉ 

LAAS-CNRS, Université de Toulouse, CNRS, ANITI, Toulouse, France

Marie-José Huguet ✉

LAAS-CNRS, Université de Toulouse, CNRS, INSA, Toulouse, France

Siham Essodaigui ✉

Renault, France

Alain Nguyen ✉

Renault, France

Abstract

Many state-of-the-art methods for combinatorial games rely on Monte Carlo Tree Search (MCTS) method, coupled with machine learning techniques, and these techniques have also recently been applied to combinatorial optimization. In this paper, we propose an efficient approach to a Travelling Salesman Problem with time windows and capacity constraints from the automotive industry. This approach combines the principles of MCTS to balance exploration and exploitation of the search space and a backtracking method to explore promising branches, and to collect relevant information on visited subtrees. This is done simply by replacing the Monte-Carlo rollouts by budget-limited runs of a DFS method. Moreover, the evaluation of the promise of a node in the Monte-Carlo search tree is key, and is a major difference with the case of games. For that purpose, we propose to evaluate a node using the marginal increase of a lower bound of the objective function, weighted with an exponential decay on the depth, in previous simulations. Finally, since the number of Monte-Carlo rollouts and hence the confidence on the evaluation is higher towards the root of the search tree, we propose to adjust the balance exploration/exploitation to the length of the branch. Our experiments show that this method clearly outperforms the best known approaches for this problem.

2012 ACM Subject Classification Mathematics of computing → Combinatoric problems; Mathematics of computing → Combinatorial optimization; Computing methodologies → Planning and scheduling; Computing methodologies → Discrete space search

Keywords and phrases Monte-Carlo Tree Search, Travelling Salesman Problem, Scheduling.

Digital Object Identifier 10.4230/LIPIcs.CP.2021.54

1 Introduction

The assembly floor of our car manufacturer partner contains several machines, each producing a certain type of components and as many machines consuming those components. The process of moving components across the workshop, from the point where they are produced to the point where they are consumed is a major bottleneck for the production rate of the plant. The resulting transportation problem can be seen as a *repetitive single vehicle pickup and delivery problem with time windows and capacity constraint*. The repetitive aspect comes from the fact that over a weekly schedule, the pickups and deliveries between the same pairs of machines is repeated at a given frequency, and for the same reason, both tasks are constrained in time. Finally, the capacity comes from the specific trolleys used by operators,



© Valentin Antuori, Emmanuel Hebrard, Marie-José Huguet, Siham Essodaigui, Alain Nguyen; licensed under Creative Commons License CC-BY 4.0

27th International Conference on Principles and Practice of Constraint Programming (CP 2021).

Editor: Laurent D. Michel; Article No. 54; pp. 54:1–54:17

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

44 which can be stacked in trains of a bounded length.

45 The method used in the industrial context is a large scale scheduling model solved
46 using local search solver. A range of approaches relying on reinforcement learning (RL)
47 were recently proposed in [2]. A simple stochastic branching policy (a linear model over
48 some problem-specific parameters) is learned via RL, and used either to guide a constraint
49 programming approach with rapid restarts, a constraint approach with limited discrepancy
50 search, or a multistart local search method. All three methods vastly outperform the
51 industrial method both on real and synthetic data sets.

52 In this paper, we introduce a new approach, combining Monte-Carlo Tree Search (MCTS)
53 with budget-limited Depth First Search (DFS). MCTS was initially designed for solving AI
54 games [6], and, over the last few years, MCTS combined with reinforcement learning and
55 deep learning has enabled a breakthrough in the resolution of many combinatorial games
56 (such as Go, with AlphaGo and AlphaGo Zero[22, 23]). Monte-Carlo Tree Search [6] offers a
57 good generic strategy to tackle combinatorial problems. The expected outcome of a subtree
58 is evaluated via Monte-Carlo simulation: starting from an open node of the search tree, a
59 complete solution is built using a randomized heuristic policy. The outcome of the rollout
60 is back-propagated to that node and all its ancestors down to the root by computing an
61 average. Then, the next node to expand is selected by traversing the search tree from the
62 root using multi-armed bandits algorithms (e.g., Upper Confidence bounds applied to Trees,
63 UCT [11]) until reaching a node that has not been expanded yet. Without requiring built-in
64 domain knowledge, Monte-Carlo rollouts provide good guidance, and the expansion phase
65 gives guarantees on the compromise between exploration and exploitation. We show that
66 in our problem, replacing the Monte-Carlo rollouts by randomized, limited-budget DFS is
67 effective.

68 Several hybridizations with combinatorial optimization frameworks have been proposed.
69 MCTS has been combined with constraint programming (CP) in [13], where the simulation
70 phase stops at first fail, and the authors do not allow backtracking. Moreover, in order to
71 allow restarts, instead of keeping an evaluation of every open node, this is done on pairs
72 variable/value, in a way inspired by the RAVE (Rapid Action Value Estimation) heuristic
73 used in Go [8]. Finally, took advantage of the fact that Gecode [7] uses copying instead
74 of trailing, to open every search node visited during a rollout. In the field of Boolean
75 satisfiability (SAT), MCTS has been combined with SAT solver [17], however, in
76 this case without including the defining characteristics (clause learning, VSIDS, ect.) of
77 modern SAT solvers. In [9], the authors propose to hybridize MCTS with local search to
78 solve the MAX-SAT problem. They use a fixed limited-budget stochastic local search in place
79 of the rollouts. Finally, in [21], the UCT algorithm has been used in mixed integer linear
80 programming (MILP), although replacing Monte-Carlo rollouts by a lower bound obtains
81 with the Linear Programming (LP) relaxation.

82 We are not aware of MCTS approaches using DFS rollouts. However, this is closely
83 related to the Hybrid Best First Search (HBFS) algorithm introduced in [1] where limited
84 DFS is interleaved with BFS, although the choice of leaf to expand is not based on the same
85 principles. Besides using DFS, we propose two adaptations of MCTS method designed to be
86 effective on our problem, but directly applicable in any combinatorial problem.

87 First, since the goal of a Monte-Carlo rollout is to evaluate a single decision, and since
88 each subsequent heuristic decision reduces the relative impact of that first decision, we argue
89 that the definition of the overall reward should reflect this form of “diminishing returns”.
90 Therefore, we propose to define the outcome of a rollout as the sum of the marginal increments
91 of the lower bound at each step, weighted by a coefficient in $]0, 1[$ that exponentially decreases

with the depth. When the coefficient tends towards 1, the outcome tends towards the overall objective value of the solution, and when it tends towards 0, the short term growth of the lower bound weigh more and more. Observe that this scheme is generic, it only requires a lower bound of the objective function which is monotonically non decreasing at each decision.

Second, in the multi-armed bandit algorithm, the tradeoff between exploration and exploitation is controled by a constant factor c for the exploration term. As the tree becomes deeper, the number of iterations of the multi-armed bandit along a branch grows. Therefore, the probability that it will deviate from the best branch so far grows exponentially with the depth of the branch. To offset this, we apply an exponential decay to the parameter c towards the root, so that the likelihood of deviating at the root decreases rapidly when the depth of the tree grows.

The paper is organized as follows. First, in Section 2 we describe the problem of routing vehicle components in car manufacturing workshops and we give a detailed overview of the standard MCTS algorithm in Section 3. Then, we present the novel aspects of our approach in Section 4. Finally, we give the specific implementation details for the considered problem in a MCTS framework in Section 5, and we report the results of extensive experiments on both industrial and synthetic data in Section 6. These experiments show that our adaptations of the MCTS method significantly outperforms previous methods, including the local search approach currently used in the industry.

2 Problem Description

The industrial assembly line consists of a set of m components to be moved across a workshop, from the point where they are produced to where they are consumed. Each component is produced and consumed by two unique machines, and it is carried from one to the other using four dedicated trolleys. Initially, there are two trolleys standing at the production point and two trolleys at the consumption point. On each side, one of them is full and the other is empty. However, the empty trolley at the production point is being filled, and the full trolley at the consumption point is being emptied. The full trolley at the production point must be brought to the consumption point before the initially full trolley there has been emptied, and symmetrically, the empty trolley at the consumption point must be brought to the production point before the initially empty trolley there has been filled. A production cycle is the time c_i taken to produce (resp. consume) component i , that is, to fill (resp. empty) a trolley. The two pickups and the two deliveries (of empty and full trolleys) described above must then be done within this time window. The end of a production cycle marks the start of the next, hence there are $n_i = \lfloor \frac{H}{c_i} \rfloor$ cycles over a time horizon H for the component i .

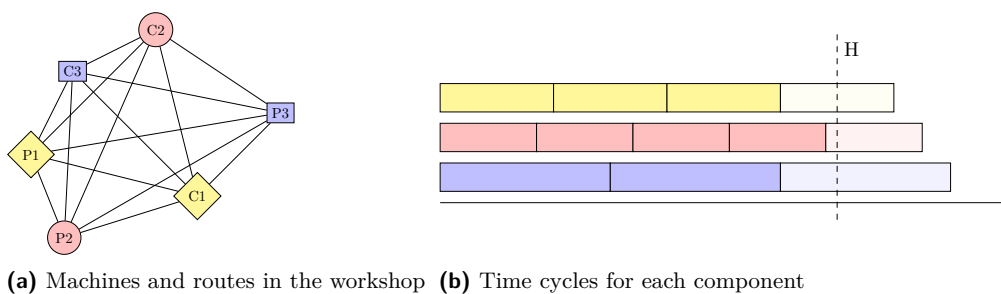


Figure 1 Illustrative example

126 The problem is illustrated on a small example in Figure 1. In this example, there are 3
 127 components having their own production and consumption machines, denoted by P_i and C_i
 128 in Figure 1(a). The lines between the machines represent the routes in the assembly line.
 129 The time cycles of each component and the time horizon (H) are given in Figure 1(b). In
 130 this example, there are 3 time cycles for the yellow component, 4 time cycles for the red
 131 component and 2 for the blue one.

132 For each component i , for each of its cycles k , there are two pickups and two deliveries:
 133 the pickup pe_i^k and delivery de_i^k of the empty trolley from the consumption area to the
 134 production one, and the pickup pf_i^k and delivery df_i^k of the full trolley from production
 135 to consumption. The processing time of an operation o is denoted pt_o and the travel time
 136 between operations o and o' is denoted $tt_{o,o'}$.

137 Let O be a set of all pickup and delivery operations with $|O| = n$. The problem is
 138 to compute a sequence $\omega : \{1, \dots, n\} \mapsto O$ of the operations O , where $\omega(j)$ is the j -th
 139 operation in the sequence, and $\chi = \omega^{-1}$ its inverse. The sequence ω must satisfy the
 140 following constraints:

141 **Routing:** For every $1 < j \leq n$, operation $\omega(j)$ must be given a start time $s_{\omega(j)}$ (and
 142 end time $e_{\omega(j)} = s_{\omega(j)} + pt_{\omega(j)}$) taking into account duration and travel time: $s_{\omega(j)} \geq$
 143 $s_{\omega(j-1)} + pt_{\omega(j-1)} + tt_{\omega(j-1),\omega(j)}$ (and $s_{\omega(1)} = 0$).

144 **Time windows:** An operation o occurring at period k for component i is given a release
 145 date $r_o = (k-1)c_i$ and a due date $d_o = kc_i$, with $r_o \leq s_o$ and $e_o \leq d_o$.

146 **Precedences:** Pickups must precede their deliveries in the same period.

$$147 \quad \chi(pf_i^k) < \chi(df_i^k) \wedge \chi(pe_i^k) < \chi(de_i^k) \quad \forall i \in [1, m] \quad \forall k \in [1, n_i] \quad (1)$$

148 **Train length:** The operator may assemble trolleys into a train (trolleys can be extracted
 149 out of the train in any order), so a pickup need not be directly followed by its delivery.

150 However, the total length of the train of trolleys must never exceed a length T_{\max} .

151 Notice that there are only two possible orderings for the four operations of a production
 152 cycle. Indeed, since the first delivery (which can be either the full or the empty trolley since
 153 they happen in parallel) and the second pickup take place at the same location, doing the
 154 second pickup before the first delivery is dominated: the train will needlessly contain both a
 155 full and an empty trolley for the same component, and this delivery will need to be done
 156 eventually and can only incur further time loss.

157 This industrial problem is a *repetitive single vehicle pickup and delivery problem with*
 158 *time windows and capacity constraint*. In this problem, the production-consumption cycles of
 159 each component entail a very particular structure: the four operations of each component
 160 must take place in the same time windows and all of these operations are repeated for every
 161 cycle. In addition, all operations are mandatory and there is no objective function for the
 162 industrial application, instead, feasibility is hard. As a result, the efficiency of the Large
 163 Neighborhood Search approaches proposed in [19] for such routing problems, are severely
 164 hampered since they rely on the length of the tour as the objective to evaluate the moves and
 165 the insertion of relaxed requests is often very constrained by the specific precedence structure.
 166 This problem was previously introduced in [2], and both exact and heuristic methods were
 167 proposed to solve it. These approaches rely on a fine tuned heuristic, and it was observed
 168 for some instances that greedy dives of the solvers were able to find a solution. The main
 169 motivation for a MCTS approach comes from this observation as the algorithm strongly rely
 170 on greedy dives, and is entirely guided by them.

3 The Monte-Carlo Tree Search Method

In this section, we give some overview of the Monte-Carlo Tree Search method, and we introduce notations that will be used in the following.

MCTS is a tree search heuristic method based on multi-armed bandit principles to guide the tree expansion and to ensure a compromise between exploration and exploitation. This method was widely studied in the context of games but also for solving optimization problems [21, 20, 16, 14, 15, 5]. For a detailed survey on the MCTS method, the reader may refer to [4]. In a nutshell, the MCTS method develops a search tree where a node corresponds to a state of a given problem, with final states being solutions. Each node is associated with a set of feasible actions leading to child nodes in the tree. The aim is to find a path from the root node to a final state maximizing a reward. The MCTS method is based on four principles:

1. a reward can be computed at each final state;
2. a simulation process, also called rollout, is used to produce a path from a given node to a final state (for instance based on random sampling);
3. a backpropagation method to update node information after each new rollouts;
4. a selection mechanism, usually based on multi-armed bandit [12], for guiding the tree expansion and insuring a compromise between exploitation (select the most promising node) and exploration (visit different parts of the tree).

Let \mathcal{A} be a set of actions. A state $\sigma \in \mathcal{A}^*$ is a sequence of actions, and $|\sigma|$ denotes its length. We note $\sigma|a$ the state reached when applying action a in state σ , $\mathcal{A}(\sigma)$ denote the set of possible actions in state σ , and $p(\sigma)$ the parent state of σ . The MCTS method stores in memory the tree \mathcal{T} it has already explored, and for every state σ , it stores the triplet: $\langle N(\sigma), Pr(\sigma), V(\sigma) \rangle$, where $N(\sigma)$ is the number of time (σ) has been visited, $Pr(\sigma)$ is the prior probability or prior preferences to choose the state σ from its parent state $p(\sigma)$, and $V(\sigma)$ is the expected value of subtrees rooted at σ , and computed by averaging the outcomes of Monte-Carlo rollouts. Notice that $Pr(\sigma)$ was introduced in the MCTS in [22] but was not in the original form of MCTS.

The algorithm iterates over the four following phases until some stopping criteria are met.

Selection

The *selection* phase begins at the root node of \mathcal{T} , and finishes when we reach a node that has not yet been explored. At each node $\sigma \in \mathcal{T}$, an action is selected according to the statistics stored in σ :

$$a^* = \arg \max_{a \in \mathcal{A}(\sigma)} \tilde{V}(\sigma|a) + c * U(\sigma|a) \quad (2)$$

where $\tilde{V}(\sigma|a)$ is the exploitation term (based on the value of node $V(\sigma|a)$), $U(\sigma|a)$ is the exploration term, and c is a parameter which represents the balance between the two terms. This process continues from the state $\sigma|a^*$ until a non-visited node is reached, i.e. a leaf of the subtree \mathcal{T} .

In adversarial games, the value V of a node is the expected outcome, e.g., 1 for a win and 0 or -1 for a loss. In the context of combinatorial optimisation, however, several definitions have been used. A first possibility is to simply store the expected objective value, although this technique entails that rollouts must be complete, even when they are suboptimal early on. In [16] and [14], the authors consider a solution whose objective value is within a factor α of the best known solution as a “win” (the effective value is in $[0, 1]$ depending on the

215 quality of solution) and all other outcomes as loss (0). The parameter α must therefore
 216 be carefully chosen, and the likelihood of a positive reward decreases when the best known
 217 solution improves. In [13], the MCTS is frequently restarted (and hence the MCTS tree lost),
 218 then the authors store the outcomes of the rollouts on variable/value pair instead. In this
 219 technique the rollouts are depth first search calls stopped on the first fail, and the expected
 220 relative failure depth is stored for each variable/value pair instantiated in the selection phase.
 221 Finally, in [21], instead of a rollout, the lower bound of the LP relaxation is backpropagated
 222 instead.

223 Observe that it is important to normalize the value V stored on the node, to make the
 224 choice of the balance exploitation/exploration parameter c more robust. A state value $\sigma|a$
 225 ending on the action a can be normalized in $[-1, 1]$ as follows:

$$226 \quad \tilde{V}(\sigma|a) = \begin{cases} 2 * \frac{V^+ - V(\sigma|a)}{V^+ - V^-} - 1 & \text{if } N(\sigma|a) > 0 \\ 0 & \text{otherwise} \end{cases} \quad (3)$$

228 Where $V^+ = \max\{V(\sigma|a) \mid a \in \mathcal{A}(\sigma), N(\sigma|a) > 0\}$ and $V^- = \min\{V(\sigma|a) \mid a \in$
 229 $\mathcal{A}(\sigma), N(\sigma|a) > 0\}$ are, respectively, the maximum and minimum values of any explored
 230 sibling state.

231 Finally, the exploration term is [22]:

$$232 \quad U(\sigma) = Pr(\sigma) \frac{\sqrt{N(p(\sigma))}}{N(\sigma) + 1} \quad (4)$$

234 The rationale is to select the action a that maximizes $\tilde{V}(\sigma|a)$ plus a bonus that decreases
 235 with each visit in order to promote exploration. The prior probability $Pr(\sigma)$ biases the initial
 236 exploration by the knowledge we have on the state. The square root term could be replaced
 237 by a logarithmic term which is often used in MCTS, without changing this rationale.

238 Expansion

239 Let σ be the node returned by the selection procedure, during the *expansion* phase, for all
 240 $a \in \mathcal{A}(\sigma)$, a child $\sigma|a$ is added to σ and initialized its visit counter $N(\sigma|a)$ with its expected
 241 objective value $V(\sigma|a)$ set to 0. If no prior probability $Pr(\sigma|a)$ is available for this state, the
 242 uniform distribution $1/|\mathcal{A}(\sigma)|$ can be used instead.

243 Simulation

244 In the simulation phase, the state obtained by the selection phase is extended to a final state
 245 τ via a Monte-Carlo rollout. In the context of combinatorial optimization the final state is a
 246 feasible solution. The rollout is typically done by random sampling of the possible actions
 247 $\mathcal{A}(\sigma)$ from state σ following a stochastic policy. For instance, one can use the probability
 248 distribution given by $Pr(\sigma|a) \mid a \in \mathcal{A}(\sigma)$. Alternatively, this can be done by any randomized
 249 greedy heuristic tailored to the problem at hand [14, 16, 20]. As mention before, hybridization
 250 with existing technologies can take place in this phase, whether it is a linear relaxation [21],
 251 a local search [9] or a call to a CP solver [13].

252 Backpropagation

253 Finally for each node σ traversed during the selection procedure, we update its statistics
 254 regarding the final state τ obtained by the simulation phase:

$$255 \quad V(\sigma) \leftarrow V(\sigma) + \frac{z(\tau, \sigma) - V(\sigma)}{N(\sigma) + 1}$$

$$256 \quad N(\sigma) \leftarrow N(\sigma) + 1$$

258 with $z(\tau, \sigma)$ the outcome of the rollout τ evaluated from node σ . The first update rule allows
 259 to maintain the average outcome of the rollouts for each node traversed during the selection
 260 step. It is possible to change the rule to only keep the best outcome found when traversing
 261 the node, instead of the average [21, 20]. The rationale is the same as minimax algorithms
 262 for games, the optimistic view is that eventually search will find the best completion of a
 263 partial solution, and therefore its expected value is closer to the best rollout than to the
 264 average of all rollouts. However, the preferred choice may depend on the standard deviation
 265 of the outcomes of rollouts under a given node, and on the ratio of the whole search tree
 266 that the algorithm will eventually explore. For this reason, for larger problems, and when
 267 the heuristic used during the rollouts is robust, the average may be better.

268 4 Tailoring Monte-Carlo Tree Search to Combinatorial Optimization

269 In this section, we introduce three modifications of standard Monte Carlo Tree Search which
 270 we empirically found beneficial in the context of optimization problems. These modifications
 271 are generic, in the sense that they hold outside of our industrial application, as long as we
 272 have a lower bound computation technique for the objective function and a depth first search
 273 procedure for the target problem.

274 4.1 Evaluation based on the objective function

275 In game playing, the outcome of a Monte-Carlo rollout may only be known when the game
 276 ends. Typically, the rollout is given a value of 1 for a win, -1 for a loss and 0 for a draw.
 277 Standard adaptations to combinatorial optimization are to normalize the objective value in a
 278 way or another as described in Section 3.

279 When simulating long branches, however, a “mistake” on a single decision along the
 280 branch can make the final outcome irrelevant. In fact, look-ahead methods often exhibit
 281 diminishing returns. For instance, it was observed in Chess that the rate of wins in self-plays
 282 between an algorithm looking $k + 1$ plies ahead versus the same algorithm looking k plies
 283 ahead declines as k grows [10]. In the case of a greedy procedure, it is therefore natural to
 284 conjecture that as the length of the branch grows, the correlation between the quality of the
 285 initial decision and the overall outcome decreases.

286 In combinatorial optimization problems, however, we usually have a lower bound on the
 287 objective that monotonically grows with every decision. Therefore, the evolution of this value
 288 can provide a better insight into the quality of an initial decision. Let $LB : \mathcal{A}^* \mapsto \mathbb{R}$ be a
 289 lower bound on sequences of actions, with $LB(\sigma)$ equals to the objective value if σ is a final
 290 state. Then, for a given node σ we propose to evaluate a state σ' reachable from σ as the
 291 sum of the marginal increment of the lower bound LB in the path from σ to σ' , weighted by
 292 an exponentially decaying coefficient γ . Hence we can define this sum recursively as follows:

$$293 \quad z(\sigma', \sigma) = \begin{cases} LB(\sigma) - LB(p(\sigma)) & \text{if } \sigma' = \sigma \\ \gamma^{|\sigma'| - |\sigma|} (LB(\sigma') - LB(p(\sigma'))) + z(p(\sigma'), \sigma) & \text{otherwise} \end{cases} \quad (5)$$

294 The evaluation of a final state τ obtained by a rollout is then simply $z(\tau, \sigma)$ and represents
 295 an upper bound of the optimal solution.

296 Algorithm 1 implements backpropagation following the reward defined in Equation 5.
 297 This algorithm takes as an input the sequence (R) of the lower bound increments given by
 298 the rollout, the node selected in the *selection* phase, and the decay rate.

■ **Algorithm 1** Backpropagation procedure

Data: R : sequence of the lower bound increments, σ : selected node, γ : decay rate

```

1 // Sum of exponentially decaying marginal increment of the lower bound
2  $val \leftarrow \sum_{i=1}^{|R|} \gamma^{i-1} R_i$ 
3 // Backpropagation until the root node
4 repeat
5    $val \leftarrow \gamma * val + LB(\sigma) - LB(p(\sigma))$ 
6    $N(\sigma) \leftarrow N(\sigma) + 1$ 
7    $V(\sigma) \leftarrow V(\sigma) + \frac{val - V(\sigma)}{N(\sigma)}$ 
8    $\sigma \leftarrow p(\sigma)$ 
9 until  $\sigma = Nil$ ;
```

299 The proposed evaluation method puts more weight on the short-term impact of a decision,
 300 wagering on it being more reliable than long term observations. For $\gamma = 1$, the score reflects
 301 the objective value $LB(\tau)$ of the rollout, whereas greater weight is put on short-term impacts
 302 when γ tends towards 0.

Moreover, the lower bound computations can be used during the expansion phase to avoid expanding into sequences whose objective value cannot be lower than the current upper bound (best known solution). Thus, a node σ' that cannot be expanded further (all potential children nodes are suboptimal) is removed from the search tree. In that case, the information is backpropagated along the branch that leads to this node, that is, each node σ containing the deleted node σ' in its subtree are updated:

$$V(\sigma) \leftarrow \frac{1}{N(\sigma) - N(\sigma')} (V(\sigma) * N(\sigma) - V(\sigma') * N(\sigma'))$$

and

$$N(\sigma) \leftarrow N(\sigma) - N(\sigma')$$

303 Indeed, all information contained in the deleted node is now irrelevant for the rest of the
 304 search as it is not in the tree anymore. Then previous iterations which have passed throughout
 305 this node should not have an impact on the future search.

306 Then, for the implementation of the proposed evaluation function, we should store $LB(\sigma)$
 307 at each node σ in addition to the triplet $\{N(\sigma), Pr(\sigma), V(\sigma)\}$.

308 A potential limit with this evaluation method is that it may skew search towards post-
 309 posing actions that greatly increase the lower bound, but must eventually be done. For
 310 instance, consider a Travelling Salesman Problem with an isolated city far away from all
 311 other cities. Rollouts where this city is visited last will be preferred to rollouts where it
 312 is visited early. Lower bounds that take into account the future decisions in a reasonable
 313 way (e.g., minimum spanning tree for the travelling salesman problem, or the preheptive
 314 relaxation in scheduling) may prevent this phenomenon since the cost of an exceptionally
 315 remote city or of an exceptionally large task would contribute to the lower bound anyways.

316 The lower bound we used in our industrial problem, however, is extremelly basic and yet
 317 this did not seem to be an issue in our experiments.

318 4.2 Dynamic Exploitation vs Exploration Balance

319 Since the tree grows deeper as search progresses, the likelihood to deviate from the best
 320 branch increases. Therefore, we propose to dynamically adapt the parameter that control
 321 the balance between exploration and exploitation, depending on the depth of the tree, in
 322 order to promote exploitation on deeper nodes. Let $td(\mathcal{T})$ be the depth of the tree \mathcal{T} , then
 323 at step t of the selection phase, the exploitation/exploration coefficient will be

$$324 \quad \beta^{td(\mathcal{T})-t} * c \quad (6)$$

325 with $\beta < 1$ a parameter. This mechanism has a similar effect as *committing to a move* at the
 326 root node. At the root $t = 0$ and thus $\beta^{td(\mathcal{T})-t} * c$ tends towards 0 when $td(\mathcal{T})$ grows, so the
 327 first decision is very unlikely to deviate from the most promising choice once the search tree
 328 has sufficiently grown. Conversely, at a leaf, this term tends towards the original value c and
 329 hence less promising – but less frequently visited – nodes will be selected more often. In the
 330 context of games, when a move is actually made, it makes sense to forget the siblings and
 331 parents of the corresponding state. In optimization, this mechanism has been implemented
 332 in several approaches in order to limit the combinatorial explosion [3, 14]. Since commits
 333 are irreversible, the algorithm is no longer complete, and budget parameters controlling such
 334 commits need to be carefully chosen. Instead, the mechanism we propose has a similar effect
 335 but in a “smooth” way: near the root, it is more likely that the best move will be chosen,
 336 however other states can still be reached.

337 4.3 Depth First Search as a rollout

338 Finally we propose to use a Depth First Search procedure instead of a randomized greedy
 339 heuristic in the simulation phase. More precisely, in order to intensify the search around
 340 promising areas, a budget is defined after a first greedy “dive” and a budget-limited DFS is
 341 performed. For this purpose, the simulation is split into three steps:

- 342 ■ The first step is a greedy randomized procedure from the selected node σ until a con-
 343 tradiction is detected. This contradiction can happen because a constraint is violated,
 344 or because the lower bound exceeds the upper bound. At this point, we define a budget
 345 for the DFS by evaluating the current state σ' . This budget will be larger if this is a
 346 promising state, and maximal if no contradiction was encountered (and hence a new
 347 upper bound was found). On the other hand, if the state σ' is not promising, then the
 348 budget will be smaller or null.
- 349 ■ The second step of the simulation is a DFS, from the state reached by the greedy procedure
 350 σ' . This search is only performed on the subtree rooted at the node σ selected in the
 351 selection phase. The DFS algorithm must be able to store the best branch discovered,
 352 that is, the best solution or the best partial sequence according to the evaluation we
 353 described previously.
- 354 ■ The third step begins when the budget is consumed (or the search is complete for
 355 the subtree rooted at the selected node σ in the selection phase). If no solution was
 356 found during the previous step, the greedy randomized procedure is used to extend the
 357 best branch found by the DFS to a complete solution which can be evaluated before
 358 backpropagation.

359 The evaluation procedures for the states and for the budget will be detailed in Section 5
 360 as their definitions depends on the considered problem.

361 **5 Adaptation to the industrial Workshop Scheduling Problem**

362 **Tree model**

363 In the search tree of the MCTS method, a state σ represents a partial sequence of operations
 364 and the set of actions correspond to the set of operations of the routing problem described
 365 in Section 2, i.e., actions are operations $\mathcal{A} = O$. In the search tree, a sequence $\sigma|a$ is the
 366 sequence σ extended by the action (operation) a . The set of possible actions $\mathcal{A}(\sigma)$ from a
 367 sequence σ contains every operation a such that the (partial) sequence $\sigma|a$ is feasible with
 368 respect to the constraints.

369 **Objective function**

Since our industrial application is a satisfaction problem (the existence of a tour without
 delay), we need to generalize it to an optimization problem to apply MCTS as described in
 Sections 3 and 4. Therefore, during the simulation phase, we relax the due date constraints
 and instead we minimize the maximum tardiness:

$$L(\sigma) = \max(0, \max_{1 \leq j \leq |\sigma|} (e_{\sigma(j)} - d_{\sigma(j)}))$$

370 Since in this case operations can finish later than their due dates, it is necessary to make the
 371 precedence constraints due to production cycles explicit:

$$372 \quad \max(\rho(df_i^{k-1}), \rho(de_i^{k-1})) < \min(\rho(pe_i^k), \rho(pf_i^k)) \quad \forall i \in [1, m] \quad \forall k \in [2, n_i] \quad (7)$$

373 Furthermore, during the expansion phase we do not add a child node that would violate
 374 a due date constraint, as our primary goal is to find a solution σ without any late job, that
 375 is, such that $L(\sigma) = 0$.

376 We use a trivial lower bound, which is at state σ the maximum tardiness $L(\sigma)$ of the
 377 associated partial sequence also taking into account tardiness of all pending operations.
 378 Pending operations are all the operations that belong to a production cycle in which at
 379 least one operation is available to extend the current sequence, ignoring the train constraint.
 380 Therefore, Equation (5) is the sum of exponentially decaying marginal increments of the
 381 maximum tardiness with a small look ahead.

382 **Heuristic**

383 For the simulation phase as well as for the probabilities of the expansion phase, we use the
 384 heuristic tuned by reinforcement learning proposed in [2]. This heuristic is stochastic and
 385 provides a probability distribution over the set of available operations for a given state. More
 386 precisely, at a given state σ , each operation $a \in \mathcal{A}(\sigma)$ is evaluated using a fitness function
 387 $f(\sigma, a)$ defined as a linear combination of four criteria: $f(\sigma, a) = \boldsymbol{\theta}^\top \boldsymbol{\lambda}(\sigma, a)$. These criteria
 388 λ_i correspond to:

- 389 1. The *emergency* of the operation: $lst(a, \sigma) - \max(r_a, e_{\sigma(|\sigma|)} + tt_{\sigma(|\sigma|), a})$, with $lst(a, \sigma)$ the
 390 latest starting time of the operation a in order to satisfy the due date constraints with
 391 respect to the operations belonging to σ and the precedences constraints;
- 392 2. The *travel/waiting time* of the operation: $\max(tt_{\sigma(|\sigma|), a}, (r_a - e_{\sigma(|\sigma|)}))$;

393 3. The (negated) *length* of the trolley;

394 4. The *type* of operation, equal to 1 for pickups and 0 for deliveries.

395 The parameter θ is set to the proposed learned values (0.251, 0.576, 0.148, 0.023). Then, a
396 **softmax** function is applied to turn the fitness evaluation into a probability distribution for
397 guiding the choice of the next node in the greedy heuristic:

$$398 \quad \forall o \in \mathcal{A}(\sigma) \quad \pi_{\theta}(o \mid \sigma) = \frac{e^{(1-f(\sigma,o))/\delta}}{\sum_{o' \in \mathcal{A}(\sigma)} e^{(1-f(\sigma,o'))/\delta}} \quad (8)$$

399 where the parameter δ controls the “greedyness” of the heuristic, that is, a “low” value for δ
400 encourages to select the best choice with high probability, whereas a more “neutral” value of
401 δ produces more randomized choices. In the experiments, we will set a value of $\delta = 0.005$ in
402 the simulation phase, and a value of $\delta = 0.1$ to initialize the prior probabilities of the new
403 nodes in the expansion phase.

404 Simulation

405 The greedy procedures before and after the DFS simply consist in taking at random the next
406 operation following the probability distribution defined by equation 8.

407 For the DFS we define a backtrack budget between 0 and \mathcal{B} , depending on when the first
408 tardiness was detected during the first dive. If the first dive finds an improving solution, then
409 the budget is maximum (\mathcal{B}), in order to find other related improving solutions. Otherwise,
410 we rely on the rank ϕ where the lower bound became positive to define the budget. Let ϕ^*
411 be the highest rank for any previous solution, the backtrack budget is then:

$$412 \quad \begin{cases} \mathcal{B} & \text{if } \phi \geq \phi^* \\ \mathcal{B} \left(\frac{\phi^* - \phi}{\phi^* - \alpha * \phi^*} \right)^2 & \text{if } \phi^* > \phi > \alpha * \phi^* \\ 0 & \text{otherwise} \end{cases} \quad (9)$$

413 with $\alpha \leq 1$, a threshold parameter.

414 During the DFS, we define a probability distribution over the children using the **softmax**
415 function of the greedy heuristic and we limit the breadth of the tree by keeping only
416 actions with a probability greater than 10^{-6} , which typically leaves all but 1 to 3 children
417 approximately. Then, those children are sorted by their probabilities, and in order to
418 randomize the DFS, a random child (again, using the same probability distribution) is
419 swapped with the first one, to be branched on first by the DFS. As instances can be very
420 large, this is sufficient to keep variety in the solutions, while removing many “bad” decisions.
421 This is also why we rely on the backtrack count instead of the fail count to define the budget,
422 as a lot of nodes may have only one child. We add a geometric restart policy in the DFS
423 step, where the search is reset to the node selected in the selection phase. The growth
424 factor is reset at each MCTS iteration. At the end, the DFS returns the longest (potentially
425 partial) sequence for which the lower bound remains null (i.e., for as the largest number of
426 operations). Then, the greedy procedure is called to extend this sequence to a complete
427 solution.

428 6 Experimental Evaluation

429 We report in this section the results of our experiments. First, we assess the respective
430 impact of using the new evaluation policy, the dynamic exploration/exploitation balance and
431 the DFS in the simulation phase. In a second part, we compare our MCTS adaptations to
432 state-of-the-art methods for this problem.

433 6.1 Experimental protocol

434 We use the same data set as in [2] composed of 120 synthetic instances. The data set is made
 435 of four categories characterized by the number of components (15 in category A, 20 in B, 25
 436 in C and 30 in D). Moreover, all of these categories are associated to three time horizons: a
 437 work shift of an operator (7 hours and 15 minutes), a work day (made up of three shifts)
 438 and a full week (6 days).

439 The number of components is highly correlated with hardness, and directly related to
 440 the branching factor in the Monte-Carlo search tree. Indeed, each node has at most two
 441 children per component (ie., from 30 children for instances of category A to 60 children for
 442 instances of category D). In addition, the depth of the search tree grows with the number of
 443 operations, that depends both on the time horizon and on the number of components. This
 444 depth varies from 450 for the “shift” schedules, up to 14500 for the “weekly” schedules.

445 We ran every method 10 times for each of the 120 instances with a timeout of 1h. All
 446 experiments were run on a cluster composed of Xeon E5-2695 v3 @ 2.30GHz processors.
 447 Our methods were implemented using in C++ and compiled with GCC-8.0. The two
 448 methods from [2] were implemented using JAVA and were run in the same conditions, and
 449 Choco-4.10 [18] for CP.

450 6.2 Impact of the MCTS adaptations

451 In the first part of the experiments, the goal is to assess the respective impact of the proposed
 452 adaptations for the MCTS method. We evaluated 6 different versions of the MCTS, adding
 453 the adaptations we propose one at a time:

- 454 ■ MCTS is the standard MCTS method without any of the proposed adaptation. This
 455 baseline method uses the value of the objective function as the result of the rollouts, and
 456 backpropagates this value through the tree to the root node.
- 457 ■ MCTS+DFS is the same algorithm as MCTS except that it uses the DFS in the simulation
 458 phase.
- 459 ■ SEDMI is the variant of MCTS that uses the sum of exponentially decaying marginal
 460 increments of the lower bound to evaluate the nodes.
- 461 ■ SEDMI+DFS adds the DFS to SEDMI for the simulation phase.
- 462 ■ SEDMI+DFS+DC extends SEDMI+DFS with the dynamic exploitation/exploration comprom-
 463 ise.
- 464 ■ SEDMI+SAT-DFS+DC is the variant of SEDMI+DFS+DC in which the upper bound on the
 465 objective function is fixed to 1 in the DFS, i.e. the DFS tries to solve the satisfaction
 466 version of the problem instead of trying to improve the global upper bound. However,
 467 the last part of the simulation still provides a complete solution via a greedy procedure,
 468 and hence this method also provides an upper bound.

469 All parameters for the proposed methods are given in Table 1. We recall that c is the
 470 exploitation/exploration tradeoff parameter. The higher value for this parameter, the more
 471 the MCTS will explore. Then, β is the decay rate for the adaptation of c , and γ is the decay
 472 rate of the evaluation function. Finally, α and \mathcal{B} are respectively the threshold parameter,
 473 and the maximum backtrack budget for the DFS. All the values for these parameters were
 474 chosen by preliminary experiments, and the chosen combination appears to give relatively
 475 good overall results.

476 The results are shown in Table 2 and 3, in which we report the number of solved runs,
 477 and the average maximum tardiness. For all the methods we consider that an instance is
 478 solved if and only if the value of the objective function is null i.e. there is no tardiness.

■ **Table 1** Parameters value

c	1
β	0.995
γ	0.9977
α	0.9
\mathcal{B}	50000
Restart (base)	100
Restart (factor)	1.2

■ **Table 2** Comparison of the MCTS adaptations

H		MCTS		MCTS+DFS		SEDMI		SEDMI+DFS		SEDMI+DFS+DC		SEDMI+SAT-DFS+DC	
		#S	L_{max}	#S	L_{max}	#S	L_{max}	#S	L_{max}	#S	L_{max}	#S	L_{max}
A	shift	100	0	100	0	100	0	100	0	100	0	100	0
	day	90	135	90	133	90	115	90	77	100	0	98	0
	week	68	1996	78	1850	70	1800	80	1839	77	1840	80	1858
B	shift	80	420	79	258	90	372	82	353	81	349	87	439
	day	50	2954	54	2959	60	2522	60	2439	63	2121	70	2134
	week	10	21070	29	20771	10	20572	32	20541	31	20355	36	20635
C	shift	49	1676	48	1708	40	1901	45	1727	40	1824	40	2012
	day	10	9503	11	9248	10	8683	26	8656	36	8747	35	9022
	week	0	64442	8	64713	0	64480	9	64584	9	64474	10	64445
D	shift	40	2154	33	2146	30	2338	33	2018	30	2304	30	2621
	day	0	13659	0	13664	0	12657	0	12723	13	12225	11	12340
	week	0	101474	0	101444	0	100533	0	100760	0	100954	0	100840
Average		41	18290	44	18241	42	17998	46	17976	48	17933	50	18029

479 Table 2 shows the performance of the different variants of the MCTS averaged by classes of
 480 instances, and by time horizons. In this table, for each method, a line corresponds to 100
 481 runs (10 instances and 10 runs for every time horizon), then the number of solved runs is
 482 a sum over these 100 runs. In Table 3 the same results are presented aggregated by time
 483 horizons, and the number of solved instances is in percentage (over the 400 runs by line and
 484 by method).

485 In these tables, we can see the benefit of using the DFS in the simulation phase. Using
 486 DFS, as expected, allows the MCTS methods to solve more instances on the *week* horizon.
 487 In fact, those instances are too large to be solved via rollouts only, and the DFS allows to
 488 intensify the search on the deepest parts of the tree, that are not explored in the MCTS.
 489 Unfortunately, the effect of the DFS is not visible on the *shift* horizon. We can also see the
 490 benefit of using the sum of exponentially decaying marginal increments as node evaluation

■ **Table 3** Results aggregated by time horizon

H	MCTS		MCTS+DFS		SEDMI		SEDMI+DFS		SEDMI+DFS+DC		SEDMI+SAT-DFS+DC	
	#S	L_{max}	#S	L_{max}	#S	L_{max}	#S	L_{max}	#S	L_{max}	#S	L_{max}
Shift	0.67	1063	0.65	1028	0.65	1153	0.65	1024	0.63	1119	0.64	1268
Day	0.38	6562	0.39	6501	0.40	5994	0.44	5974	0.53	5773	0.54	5874
Week	0.20	47245	0.29	47195	0.20	46846	0.30	46931	0.29	46906	0.32	46944

491 on *day* and *week* horizons in terms of objective value. However, this adaptation slightly
 492 degrades the performance on shorter horizon meaning that this time horizon is too short to
 493 take advantage of this mechanism. Overall, the combination of both mechanisms outperforms
 494 the two versions with only one of these mechanisms. Finally, the effect of the dynamic
 495 compromise can be seen on the *day* horizon. This time horizon is small, but not enough for
 496 the MCTS to advance deep enough in the search tree to find solutions. This mechanism
 497 forces the MCTS to explore the tree deeper and faster, and as a results, to improve the
 498 number of solved instances.

499 6.3 Comparison with previous methods

500 For the second part of the experiments, we compare the two best MCTS methods, namely
 501 SEDMI+DFS+DC (the method leading to the lowest objective function) and SEDMI+SAT-DFS+DC
 502 (the method with the highest number of solved instances) to the two best methods introduced
 503 in [2], that are both based on the stochastic branching policy described in Section 5:

- 504 ■ CP: a constraint programming approach with rapid restarts. This method solves the
 505 satisfaction version of the problem. As a result, it is slightly better for finding solutions
 506 without tardiness.
- 507 ■ GRASP: a multi-start local search procedure. This method considers the optimization
 508 problem with relaxed due dates as in the MCTS methods, hence we can compare the
 509 overall tardiness.

■ **Table 4** Comparison with previous methods

H		CP		GRASP		SEDMI+DFS+DC		SEDMI+SAT-DFS+DC	
		#S	#S	L_{max}	#S	L_{max}	#S	L_{max}	
A	shift	90	90	10	100	0	100	0	
	day	90	90	193	100	0	98	0	
	week	80	70	2433	77	1840	80	1858	
B	shift	60	60	467	81	349	87	439	
	day	52	46	3218	63	2121	70	2134	
	week	35	10	26915	31	20355	36	20635	
C	shift	40	40	1941	40	1824	40	2012	
	day	10	10	9498	36	8747	35	9022	
	week	10	0	71104	9	64474	10	64445	
D	shift	19	16	2677	30	2304	30	2621	
	day	0	0	13994	13	12225	11	12340	
	week	0	0	107186	0	100954	0	100840	
Average		40.5	36	19969	48	17933	50	18029	

510 The results, given in Table 4, show that overall, the proposed MCTS adaptations
 511 outperform the CP and the local search approaches on both criteria: the number of solved
 512 instances, and the maximum tardiness. More precisely, the dominance is clear for horizons
 513 *shift* and *day* in terms of number of instances solved, but we can see that our method does
 514 not outperform the CP model on the *week* horizon. Finally, between the CP approach and
 515 the SEDMI+SAT-DFS+DC variant, there is a difference of 9.5% of instances solved in favor of
 516 the latter. There is still half of the instances that are not solved to optimality. However, the
 517 instances of the data set were randomly generated without a guarantee of satisfiability, and,
 518 we believe that the majority of unsolved instances are not satisfiable (especially for the week
 519 horizon).

7 Conclusion

In this paper, we have presented and applied several variants of the Monte Carlo Tree Search method to solve a repetitive single vehicle pickup and delivery problem with time windows and capacity constraint, issuing from car manufacturing assembly lines. We defined a way of evaluating the rollouts based on the growth of the lower bound of the objective function. We also proposed an adaptation of the balance parameter between exploitation and exploration in order to be able to solve larger instances. Moreover, we proposed an hybridization of Monte Carlo Tree Search with Depth First Search used during the simulation phase. The experimental evaluation demonstrates that these proposals allow us to outperform previous approaches on the considered problem, and show the benefit of our contributions.

These three proposals, although well suited to a dedicated problem, are generic. The next step is then to demonstrate the genericity of these Monte Carlo Tree Search variants by considering their application to other combinatorial optimization problems. We also plan to integrate our MCTS method in existing constraint programming solvers to take advantage of their search tree exploration in the Depth First Search part, further reinforcing the hybrid nature of the approach. Finally, we would like to explore further the learning aspects of the method. Indeed, in the simulation phase, we are repeatedly dealing with similar subproblems in different part of the tree, and the policy used in a subtree could be adjusted after each iteration in order to have different policies adapted to different parts of the tree search.

References

- 1 David Allouche, Simon de Givry, George Katsirelos, Thomas Schiex, and Matthias Zytnicki. Anytime hybrid best-first search with tree decomposition for weighted CSP. In *Proceedings of the 21st International Conference on Principles and Practice of Constraint Programming (CP)*, pages 12–29, 2015. doi:10.1007/978-3-319-23219-5_2.
- 2 Valentin Antuori, Emmanuel Hebrard, Marie-José Huguet, Siham Essodaigui, and Alain Nguyen. Leveraging Reinforcement Learning, Constraint Programming and Local Search: A Case Study in Car Manufacturing. In *Proceedings of the 26th International Conference on Principles and Practice of Constraint Programming (CP)*, pages 657–672, 2020. doi:10.1007/978-3-030-58475-7_38.
- 3 Dimitris Bertsimas, J. Daniel Griffith, Vishal Gupta, Mykel J. Kochenderfer, and Velibor V. Misić. A comparison of Monte Carlo tree search and rolling horizon optimization for large-scale dynamic resource allocation problems. *European Journal of Operational Research*, 263(2):664–678, 2017. doi:10.1016/j.ejor.2017.05.032.
- 4 Cameron Browne, Edward Jack Powley, Daniel Whitehouse, Simon M. Lucas, Peter I. Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez Liebana, Spyridon Samothrakis, and Simon Colton. A Survey of Monte Carlo Tree Search Methods. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(1):1–43, 2012. doi:10.1109/TCIAIG.2012.2186810.
- 5 Guillaume Chaslot, Steven Jong, Jahn-Takeshi Saito, and Jos Uiterwijk. Monte-Carlo Tree Search in Production Management Problems. In *Proceedings of the 18th Belgium-Netherlands Conference on Artificial Intelligence (BNAIC)*, pages 91–98, 01 2006.
- 6 Rémi Coulom. Efficient Selectivity and Backup Operators in Monte-Carlo Tree Search. In *Proceedings of the 5th International Conference on Computers and Games (CG)*, pages 72–83, 2006. doi:10.1007/978-3-540-75538-8_7.
- 7 Gecode Team. Gecode: Generic constraint development environment, 2006. Available from <http://www.gecode.org>.
- 8 Sylvain Gelly and David Silver. Combining online and offline knowledge in UCT. In *Proceedings of the 24th International Conference on Machine Learning (ICML)*, pages 273–280, 2007. doi:10.1145/1273496.1273531.

- 568 9 Jack Goffinet and Raghuram Ramanujan. Monte-Carlo Tree Search for the Maximum Satisfiability Problem. In *Proceedings of the 22nd International Conference on Principles and Practice of Constraint Programming (CP)*, pages 251–267, 2016. doi:10.1007/978-3-319-44953-1_17.
- 570
- 571 10 Ernst A. Heinz. New Self-Play Results in Computer Chess. In *Proceedings of the Second International Conference on Computers and Games (CG)*, pages 262–276, 2000. doi:10.1007/3-540-45579-5_18.
- 572
- 573
- 574 11 Levente Kocsis and Csaba Szepesvári. Bandit Based Monte-Carlo Planning. In *Proceedings of the 17th European Conference on Machine Learning (ECML)*, pages 282–293, 2006. doi:10.1007/11871842_29.
- 575
- 576
- 577 12 Levente Kocsis and Csaba Szepesvári. Bandit based monte-carlo planning. In *Proceedings of the 17th European Conference on Machine Learning (ECML)*, ECML'06, page 282–293, Berlin, Heidelberg, 2006. Springer-Verlag. doi:10.1007/11871842_29.
- 578
- 579
- 580 13 Manuel Loth, Michèle Sebag, Youssef Hamadi, and Marc Schoenauer. Bandit-Based Search for Constraint Programming. In *Proceedings of the 19th International Conference on Principles and Practice of Constraint Programming (CP)*, pages 464–480, 2013. doi:10.1007/978-3-642-40627-0_36.
- 581
- 582
- 583
- 584 14 Jacek Mandziuk and Cezary Nejman. UCT-Based Approach to Capacitated Vehicle Routing Problem. In *Proceedings of the 14th International Conference on Artificial Intelligence and Soft Computing (ICAISC)*, pages 679–690, 2015. doi:10.1007/978-3-319-19369-4_60.
- 585
- 586
- 587 15 Shimpei Matsumoto, Noriaki Hirose, Kyohei Itonaga, Nobuyuki Ueno, and Hiroaki Ishii. Monte-carlo tree search for a reentrant scheduling problem. In *Proceedings of the 40th International Conference on Computers Industrial Engineering (CIE)*, pages 1–6, 2010. doi:10.1109/ICCIE.2010.5668320.
- 588
- 589
- 590
- 591 16 Minh Anh Nguyen, Kazushi Sano, and Vu Tu Tran. A monte carlo tree search for traveling salesman problem with drone. *Asian Transport Studies*, 6:100028, 2020. URL: <http://www.sciencedirect.com/science/article/pii/S2185556020300286>, doi:<https://doi.org/10.1016/j.eastsj.2020.100028>.
- 592
- 593
- 594
- 595 17 Alessandro Previti, Raghuram Ramanujan, Marco Schaerf, and Bart Selman. Monte-Carlo Style UCT Search for Boolean Satisfiability. In *Proceedings of the 12th International Conference of the Italian Association for Artificial Intelligence (AI*IA)*, pages 177–188, 2011. doi:10.1007/978-3-642-23954-0_18.
- 596
- 597
- 598
- 599 18 Charles Prud'homme, Jean-Guillaume Fages, and Xavier Lorca. *Choco Solver Documentation*. TASC, INRIA Rennes, LINA CNRS UMR 6241, COSLING S.A.S., 2016. URL: <http://www.choco-solver.org>.
- 600
- 601
- 602 19 Stefan Ropke and David Pisinger. An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. *Transportation Science*, 40(4):455–472, 2006. doi:10.1287/trsc.1050.0135.
- 603
- 604
- 605 20 Thomas Philip Runarsson, Marc Schoenauer, and Michèle Sebag. Pilot, Rollout and Monte Carlo Tree Search Methods for Job Shop Scheduling. In *Proceedings of the 6th International Conference on Learning and Intelligent Optimization (LION)*, pages 160–174, 2012. doi:10.1007/978-3-642-34413-8_12.
- 606
- 607
- 608
- 609 21 Ashish Sabharwal, Horst Samulowitz, and Chandra Reddy. Guiding combinatorial optimization with UCT. In *Proceedings of the 9th International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR)*, pages 356–361, 2012. doi:10.1007/978-3-642-29828-8_23.
- 610
- 611
- 612
- 613 22 David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Vedavyas Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy P. Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016. doi:10.1038/nature16961.
- 614
- 615
- 616
- 617
- 618

- 619 23 David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur
620 Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy P.
621 Lillicrap, Fan Hui, Laurent Sifre, George van den Driessche, Thore Graepel, and Demis
622 Hassabis. Mastering the game of go without human knowledge. *Nature*, 550(7676):354–359,
623 2017. doi:10.1038/nature24270.