Knowledge-Based Systems 225 (2021) 107067

Contents lists available at ScienceDirect

Knowledge-Based Systems

journal homepage: www.elsevier.com/locate/knosys

The hierarchical task network planning method based on Monte Carlo Tree Search

Tianhao Shao, Hongjun Zhang, Kai Cheng [*], Ke Zhang, Lin Bie

Army Enginnering University of PLA, Nanjing, PR China

A R T I C L E I N F O

Article history:
Received 9 June 2020
Received in revised form 21 November 2020
Accepted 20 April 2021
Available online 26 April 2021

Keywords:
Hierarchical task network
Monte Carlo Tree Search
Intelligent planning
Optimal solution
Simulation

A B S T R A C T

Since the hierarchical task network (HTN) planning depends on the domain knowledge of the problem, the planning result relies on the writing order of the decomposition method. Besides, the solution obtained by planning is usually a general feasible solution, which means there are shortcomings in the ability of finding the optimal solution. In order to reduce the dependence of HTN planning on domain knowledge and obtain a better planning solution, Pyhop-m, an HTN planning algorithm based on Monte Carlo Tree Search(MCTS) is proposed. In the planning process, a planning tree is built by MCTS to guide the HTN planner to choose the best decomposition method. Experiments illustrates that whether in the static or dynamic environment, Pyhop-m is superior to the existing Pyhop and heuristic-based Pyhop-h in plan length, planning success rate and optimal solution rate. Under the 95% confidence level, the confidence intervals of Pyhop-m algorithm to achieve the planning success rate and the optimal solution rate in the dynamic environment are [75.82%,89.18%] and [88.67%,93.95%], which are significantly higher than those of Pyhop-h with [58.19%,77.81%] and [69.91%,80.69%], respectively. Moreover, it can solve the planning problem with uncertain action executions by repeatedly simulating and evaluating the leaf nodes of the planning tree. It can be concluded that Pyhop-m can not only make the planning result independent of the writing order of the decomposition methods, but also search out the global optimal solution.

1. Introductions

As a kind of intelligent planning technology, the HTN has been widely used in practical planning problems such as emergency plan formulation [1], real-time strategy game [2], robot control [3] and automatic combination of web services [4] because the planning process of HTN is similar to the thinking process of human problem solving. The HTN seeks a feasible solution to complete the task by decomposing the task and dissolving the conflict. The basic idea is to extract the special knowledge in the field of planning, which is used to recursively decompose the complex and abstract tasks in the field into smaller and smaller subtasks, until the decomposed subtasks can be directly completed by specific planning actions. The planned sequence of subtasks is called a course of actions (COA).

Compared with classical planning, the HTN planning has domain-specific knowledge as inputs, which makes it be better to model and expand to solve more complicated and special problems. However, this also leads to a problem with HTN planning: the planning has a strong dependence on domain knowledge, and for the same problem, writing domain knowledge in different orders may result in different planning results. This problem also increases the burden of domain experts writing domain knowledge [5].

HTN planning needs to be completed with the help of a planner, which is a component utilized to decompose and combine task networks in the planning space and is the core of HTN planning. Since HTN was proposed in 1975 [6], there have been various HTN planners, the most classic of which is the simple hierarchical ordered planner proposed by the team of Dana—SHOP and its successor SHOP2 [7–9]. SHOP and SHOP2 use an ordered task decomposition algorithm to plan tasks, that is, the COA generated during planning is consistent with the COA during execution. Whereas, due to the sequence and simplicity of the algorithm, it brings another problem in HTN planning: a feasible solution rather than the optimal solution of the problem tends to be obtained in planning.

In summary, how to reduce the correlation between planning results and domain knowledge and find a global optimal planning solution at the same time has been the focus of HTN planning research. In this study, we use the idea of MCTS to select the best decomposition method for HTN compound tasks by means of simulation. Our contributions are as follows:

[*] Corresponding author.
E-mail addresses: 296749641@qq.com (T. Shao), jsnjzhj@263.net (H. Zhang), chengkai911@msn.com (K. Cheng), 2387303531@qq.com (K. Zhang), insomniaeggy@126.com (L. Bie).

https://doi.org/10.1016/j.knosys.2021.107067
0950-7051/© 2021 Elsevier B.V. All rights reserved.

- The problem that the planning solution depends on the writing order of methods in HTN planning is studied in depth. A method selection algorithm MCTS-HTN based on MCTS is proposed, and via the four steps of Selection, Expansion, Evaluation and Backpropagation in MCTS, a suitable method for the compound tasks in HTN is found, hence the influence of methods writing sequence on planning solution is eliminated and writing burdens of domain experts are alleviated.
- The optimal solution problem in HTN planning is researched in depth. The CalValue function is designed for the leaf node evaluation of MCTS as the objective function of the optimization problem. Different CalValue functions can be utilized to calculate and evaluate according to the preferences of users, and different optimal schemes are able to be generated for different user preferences.
- Combined with MCTS-HTN, a planning algorithm, Pyhop-m, is proposed and implemented in Python language based on SHOP planner. Experimental results on classical path planning problems reveal that the proposed algorithm is superior to the existing Pyhop algorithm and Pyhop-h algorithm based on heuristic functions in terms of plan length, planning success rate and optimal solution rate in both static and dynamic environments. Besides, our method can solve the planning problem with uncertain action execution by repeatedly simulating and evaluating the leaf nodes of the planning tree.

The remainder of this paper is organized as follows: Section 2 briefly introduces the basic theory of HTN planning and the specific problems for improvement. Section 3 proposes HTH planning based on MCTS, where the definition of HTN is given and expanded in Section 3.1, Section 3.2 proposes MCTS-HTN algorithm based on MCTS, which can find out the best decomposition method for compound tasks, and Section 3.3 proposes SHOP-m algorithm by combining MCTS-HTN with classic HTN planner SHOP. Section 4 implements SHOP-m in Python—the Pyhop-m algorithm and compares it with the existing Pyhop algorithm and Pyhop-h algorithm based on heuristic search through four experiments. Section 5 summarizes the work of this paper and gives an outlook for future research.

2. Related works

The HTN planning model can be summarized as three spaces: state space, task space and knowledge space and two systems: planning system and execution system [10]. The initial state, initial task network and domain knowledge generated by the above three spaces are utilized as the inputs of HTN planning. By means of the planning in the planning system, the COA composed of primitive tasks is outputted, and it is finally executed by the execution system in the specific environment. The planner in the planning system recursively decomposes complex tasks into smaller and smaller subtasks by using a decomposition method until all the subtasks can be directly executed by actions of the domain knowledge. In HTN, the tasks that need to continue to be decomposed are called compound tasks, and the tasks which can be directly executed are referred to as primitive tasks [11].

The HTN planner will randomly select the tasks that meet the constraints in the task network to decompose or execute, and for each compound task, it will randomly choose the decomposition method that satisfy its prerequisites to perform task decomposition. Two uncertainties in the choice of algorithms result in that HTN often has diverse options at each moment of planning, which increases the dependence of planning on domain knowledge. For this problem, the family of SHOP planners adopt a sequential selection scheme, that is, they select the next task to decompose or execute according to the task order in the initial task network and match the decomposition method of the compound task according to the writing order in the domain knowledge.

Although the sequence planner greatly reduces the uncertainty in planning, it also brings a problem in HTN planning: the planning has a strong dependence on domain knowledge. First of all, the sequential selection of decomposition methods makes the writing sequence of methods particularly significant. If the order of a decomposition method that meets the prerequisites of the compound task is lower in the task decomposition methods list, and none of the decomposition methods at the front meets the prerequisites of the task decomposition, multiple recursions and backtrackings will occur according to the rules of sequential decomposition. Therefore, writing domain knowledge in a different order may result in different planning results. In order to make the domain knowledge of HTN more general, some researchers combine HTN planning with classic planning and use HTN when they need, otherwise they can return to classic planning. For example, Duet Planner [12] combines the HTN domain knowledge in SHOP2 with the local search technique in the planning graph [13] to reduce the degree of dependence on HTN domain knowledge in planning. Other researchers applied similar ideas to heuristic search in classical planning to enhance the planning capabilities of the HTN planner. For instance, H2O utilizes domain-independent heuristic search to estimate how close the effect of the decomposition method is to the current state [14]; Pyhop-h utilizes domain-independent heuristic search to estimate how close the effect of the decomposition method is to the final state [15]; POPR uses a heuristic method to delete certain constraints in the plan, and plan repair template is designed to plan the problem [16]. On the other hand, similar tasks with different purposes in the same field may have different preferences for decomposition methods. For instance, in the classic logistics field, if the goods should be transported between two distant places, when the goal is transportation time, the decomposition method of “transport by plane” should be chosen first; when the goal is transportation cost, the decomposition method “transport by train” should be selected first. The known landmarks in classical planning were used in [17,18] to judge whether the current decomposition is better than the previous one by setting a “minimum” method, so as to choose an optimal decomposition method for compound tasks.

Planning with the help of heuristic functions and landmarks can reduce the dependence of planning on domain knowledge to a great extent, however, these methods only consider the optima at the planning moment without overall planning of tasks from a global perspective. This leads to another problem in HTN planning: the COA obtained by planning is not the optimal planning solution. When the planner gets a set of COA that can solve all the tasks, it will stop planning. This set of COA is only one feasible solution by far that can meet the task conditions. If the planning continues, the better COA may keep being produced. “The better” here can be fewer steps, fewer resources, or faster tasks completion and so on. In order to enable HTN to output better feasible solutions, the SHOP2 planner adds the cost value (or utility value) after each decomposition method is selected. The planner will choose the least cost (or most effective) method among the alternative methods to decompose the current task to get a better solution of HTN [9]. [19] constructed a new heuristic HTN planning method, F-HTN, which uses Simple Temporal Networks to express complex time constraints in planning, thus constructing a new search direction and obtaining a better solution to HTN. Binding the HTN problem to a certain depth of decomposition, [20] translated the problem into a propositional formula to get the optima. [21,22] describes how to use any classic heuristic algorithm in HTN, which promotes the development of heuristic search in HTN planning. However, these methods also have a common defect, which is that the solution obtained by the planning is only the optimal solution when we observe one step forward, which means that it is the optimal solution under the greedy condition, and such a planning solution does not represent the global optimal solution sometimes.

In order to separate the HTN planning results from the domain knowledge and obtain the global optimal planning solution, this paper focuses on the method of adding MCTS to HTN planning to improve the performance of HTN planning, help the HTN planner output the optimal COA, and separate the planning from the method sequence simultaneously. Dana and Patra used the idea of MCTS for integrating planning and acting. They replaced the description model in planning with an operational model to address inconsistencies in planning and acting, thereby separating the two processes of planning and acting [23,24]. Using the idea of MCTS in the planning structure to obtain a COA that is observed several steps later and submit it to the acting structure to finally achieve the task planning in a dynamic environment [25]. The biggest difference between the operational model and the traditional HTN planning model is the method of the former has a more complicated branch structure. The complex method coding structure can solve most uncertain planning problems such as the problem of robot door opening proposed in [26]. Nonetheless, it also brings difficulties in writing domain knowledge. The original HTN domain knowledge cannot be directly transplanted and the method needs to be modified. Applying operational model to expand HTN cores to the HTN actor, and the operational model is utilized to perform task planning during the acting. The disadvantage of operational model is that its ideas are different from the traditional HTN, so the existing domain knowledge cannot be reused and expanded. On the other hand, MCTS was used in commander behavior modeling in [27] to construct a search tree through the abstraction of states and actions, and HTN planning was utilized to guide the search process and facilitate the search efficiency. The core of this method lies in tree search rather than HTN. Hence, the disadvantage is that the extension of the tree is completely controlled by HTN, which makes the behavior modeling process overly relying on HTN domain knowledge. Based on the traditional HTN structure, this paper takes the HTN planner as the core, takes MCTS as an auxiliary planning tool and performs actions after planning. This modes has good portability to the existing HTN domain knowledge and does not require further branch expansion of the domain knowledge. At the same time, it can be easily combined with the existing HTN planner (such as the family of SHOP planner), then reduce the dependence of planning on domain knowledge and obtain the global optimal planning solutions.

Our main work still uses the traditional HTN planning framework. Fig. 1 reveals the schematic diagram of the HTN overall planning with MCTS. When a decomposition method needs to be selected for the compound task t, MCTS takes the state S in the current plan as input, and the simulated output result is the decomposition method order of task t. Then HTN chooses the best decomposition method of task t according to the order. Our method can also be applied to dynamic environments. Each planning obtains a COA consisting of N actions, which is executed in the actual environment. If the external environment changes and the execution premise of the next action o is not met (or the task goal changes), or all actions in COA are performed, the current unfinished tasks and the current actual world state will be taken as inputs, and the next N actions are replanned.

Our work is based on the Python version of the SHOP planner-Pyhop and the Pyhop-h algorithm for enhancing SHOP with domain-independent heuristic search in [15]. Based on MCTS, a new task planning algorithm Pyhop-m is proposed, and it shows that the planning results obtained by Pyhop-m are better than Pyhop and Pyhop-h under different planning conditions.

3. The HTN planning based on MCTS

This section first reveals the definitions of HTN and extended Monte Carlo tree nodes. Secondly, the MCTS-HTN algorithm used by MCTS in HTN planning is proposed. Finally, combining MCTS-HTN with the classic SHOP planning algorithm, a new HTN planning algorithm—SHOP-m algorithm is constructed. SHOP-m can utilize the method of Monte Carlo simulation to select the global optimal decomposition method for the tasks that need to be decomposed at the moment. Whether in a static environment or a dynamic one, it can reduce the dependence of planning results on domain knowledge and obtain real-time optimal planning solution.

3.1. Formalization

The formal description of HTN has different forms of expression in various studies, but it generally includes three parts of state space, task network, domain knowledge, and planning problems composed of the above three parts.

Definition 1 (*State Space*)**.** The description of the state space by the HTN planner is usually completed through predicate logic. Predicate expression p consists of items $\tau_1, \tau_2, \dots, \tau_n$, where the items can be a constant or a variable. Predicates that do not contain variables are called atomic predicates. There are two types of predicates: true and false. Define S as state space, P as the set of predicate expressions, and S is the conjunction expression of 2^P elements of P:

$$S = p_1 \wedge p_2 \wedge \cdots \wedge p_{2^P}$$

In reality planning, the state space is often not discrete but continuous. In this case, in addition to the two-valued predicate expressions in S, there can be also continuous equations, which is conducive to the realization of coding. For example, Both SHOP and SHOP2 take this form.

Definition 2 (*Task Network*)**.** The tasks in HTN are composed of primitive task A and compound task C. The task network T_n is defined as a two-tuple:

$$T_n = \langle T, \psi \rangle$$

where T represents the task, which includes primitive task A and compound task C. Primitive tasks refer to the tasks that can be directly executed by actions, and compound tasks are the tasks that can be decomposed into subtasks by decomposition methods, where subtasks can be primitive tasks or compound tasks. ψ represents the constraint relationship between tasks and specifies the constraints on T that must be met during planning and execution.

Definition 3 (*Domain Knowledge*)**.** The domain knowledge D in HTN consists of HTN operations and methods, which is the core of HTN coding and planning. It is defined as:

$$D = \langle O, M \rangle$$

where O and M represent a limited set of operations and a set of decomposition methods respectively.

Each operation o in the operation set O is defined as a triple:

$$o = \langle a, pre\,(a)\,, eff(a) \rangle$$

where a represents a primitive task that can be performed by the operation, $pre\,(a)\,\epsilon P$ refers to the prerequisites for the operation, $eff\,(a)\,\epsilon P$ represents the effect of the operation on the state space which includes two subsets $eff\,(a)^+$ and $eff\,(a)^-$. They respectively represent the set of predicate expressions which are added

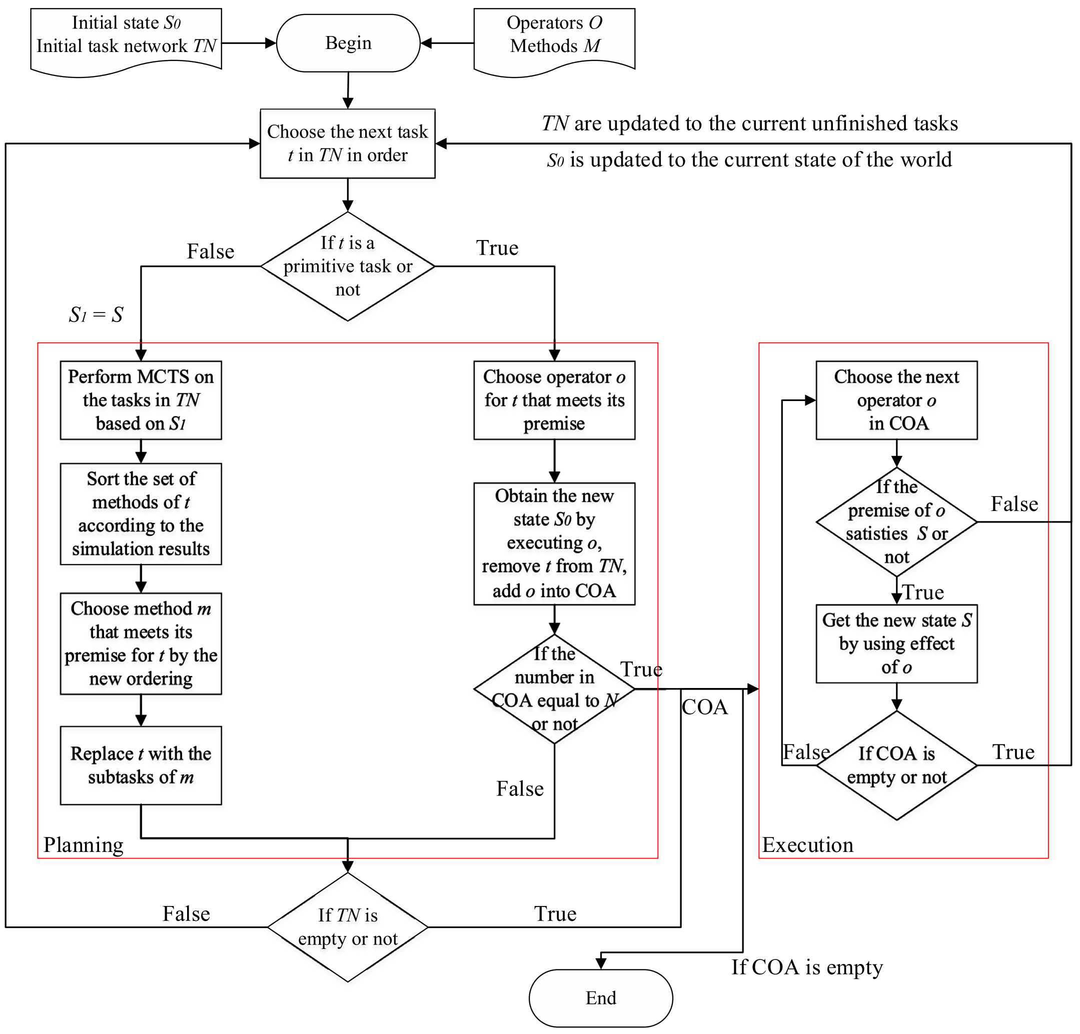
Initial state S0
Initial task network TN
Begin
Operators O
Methods M
Choose the next task t in TN in order
TN are updated to the current unfinished tasks
S0 is updated to the current state of the world
If t is a primitive task or not
False
True
S1 = S
Perform MCTS on the tasks in TN based on S1
Sort the set of methods of t according to the simulation results
Choose method m that meets its premise for t by the new ordering
Replace t with the subtasks of m
Choose operator o for t that meets its premise
Obtain the new state S0 by executing o, remove t from TN, add o into COA
If the number in COA equal to N or not
True
False
COA
Planning
Choose the next operator o in COA
If the premise of o satisties S or not
False
True
Get the new state S by using effect of o
If COA is empty or not
False
True
Execution
If TN is empty or not
False
True
If COA is empty
End

Fig. 1. Overall planning of HTN with MCTS participation.

or deleted to the state space when the operation is executed. The instantiation of operation is called action, and action is referred to operation in this paper.

Each method m in the decomposition method set M is defined as a triple:

$$m = \langle c, pre\left(c\right), T_n(c)\rangle$$

where c represents a compound task that can be decomposed by this method, $pre\left(c\right) \epsilon P$ indicates the prerequisites for the execution of this decomposition, $T_n(c)$ refers a network of subtasks decomposed from compound tasks c, including task set and constraint relationship set.

Definition 4 (*Planning Problem*)**.** The HTN planning problem P_b is a triple consisting of initial state, initial task network, and domain knowledge:

$$P_b = \langle S_0, T_{n0}, D\rangle$$

Each HTN planning problem has a COA that can solve all tasks in T_{n0}, which is called plan π. The plan consists of a collection of actions: $\pi = (o_1, o_2, \ldots, o_n)$. $\pi \neq \varnothing$ means that P_b has a solution and according to the actions in π, all the tasks in T_{n0} can be solved.

Combining MCTS with HTN planning, the construction of a Monte Carlo tree structure with HTN planning elements is required.

Definition 5 (*Planning Node*)**.** The planning node N_d consists of five attributes: *type*, *children*, *parent*, *num* and Q, so N_d is defined as a quintuple:

$$N_d = \langle tp, child, pr, N, Q\rangle$$

where *tp* represents the type of node, and the values are “task”, “method” or “action”; *child* is used to store the child nodes of the node; *pr* refers to the parent node of this node, if the node is the root, the parent node is empty; N is the simulation times of the node. In each simulation, if the node appears during the simulation, the value of N is increased by 1; Q refers to the value of the node. N and Q are the key elements in MCTS. The Monte Carlo simulation process is to continuously modify the N value and Q value according to each simulation result until it finally reaches the convergence state. Then, the best Monte Carlo search tree is formed according to the final N and Q.

3.2. MCTS-HTN algorithm

MCTS is a simulation method based on probabilistic and statistical theoretical methods. The search tree is constructed through selection, expansion, evaluation and backpropagation. When the number of simulations is sufficient enough, the simulation results will converge to the optimal solution, which has been proved in [28]. MCTS has been applied to many fields such as finance, medicine and games. Especially, the application of guiding AlphaGo to search for the best place in the game of Go illustrated that it has broad application prospects in the field of artificial intelligence [29,30]. In this study, the MCTS method is utilized to build an optimal planning tree via a four-step process: selection, expansion, evaluation and backpropagation, which is to guide the HTN planner to select the optimal decomposition method for the tasks that require decomposition currently [31].

When the HTN planner needs to select the decomposition method for a task, it will call the MCTS-HTN function, taking the current state copy *S_c*, the current task network *TN*, the operation set *O* and method set *M* as parameters. The MCTS-HTN function is the entry function of the root task, and several simulations will be performed (line 2). During each simulation, all the tasks in *TN* will be searched. Different types of nodes can be created depending on the task types, which is done by calling the Exps_TreeNode class. Essentially, a new instance is created with parameters that assign the "name" and "type" properties of the class (the usage of creating a method node is similar in the following sections). Then, the MCTS-Task and MCTS-OperatorEvlt functions are called to search down according to the type of created node (lines 3 to 10). When the search is complete, *Q* and *N* of root task node are updated (lines 11 to 12). When all the simulations are finished, the function sorts *Q* and *N* of all the tasks in *TN* in descending order and returns their values (lines 15 to 16).

```
Procedure MCTS-HTN(S_c,TN,O,M)
1:  Q←NULL;N←NULL
2:  for i in range(SimulateNum)
3:    for t in TN
4:      if t is a compound task then
5:        node←Exps_TreeNode(t,'task')
6:        S_c,index,Q_t←MCTS-Task(S_c,t,O,M,node,1)
7:      else if t is a primitive task
8:        node←Exps_TreeNode(t,'operator')
9:        S_c,Q_t←MCTS-OperatorEvlt (S_c,t,O,M,node,1)
10:     endif
11:     if t is the first task in TN
12:       Q[index]←Q_t;N[index] + 1
13:   endfor
14: endfor
15: sort Q and N in descending order
16: return Q,N
End MCTS-HTN
```

The MCTS-Task function is used to simulate the compound task node. If all the methods of this task node have been simulated, then the Selection_Med function is utilized to select the method that requires further simulation according to the rules in the MCTS. Otherwise, select the next method that has not been simulated in order and create the corresponding method node (lines 2 to 9). Afterwards, the MCTS-Method function is called to conduct a further simulated search of the method node selected by the task. After the search is completed, update *Q* and *N* values of the node and return (lines 10 to 13).

```
Procedure MCTS-Task(S_c,t,O,M,node,d)
1:  Relevant←M[t]
2:  if all method in Relevant have been simulated then
3:    index←Selection_Med(node)
4:    med←Relevant[index]
5:    newnode←node.children[index]
6:  else med←next method that has not been simulated
7:    newnode←Exps_TreeNode(med, 'method')
8:    add newnode into node.childrem
9:  endif
10: S_c,Q_m←MCTS-Method(S_c,t,med,O,M,newnode,d)
11: node.N + 1
12: node.Q←the average of the Q values of nodes in node.children
13: return S_c,index,node.Q
End MCTS-Task
```

The MCTS-Method function is used to simulate the method node. First, the corresponding task is decomposed. If it cannot be decomposed, the *Q* value of the current node is set to 0 (lines 2 to 6). Otherwise, the subtasks obtained by decomposition are sorted and search forward again according to the form in the MCTS-HTN function (lines 7 to 21). After the search is complete, update the *Q* and *N* of the current method node and return (lines 22 to 24). It should be emphasized that the first line of the function determines whether the current depth *d* exceeds the maximum depth of the search, if so, the value *Q_d* of *Q* corresponding to the maximum depth is returned, which can effectively reduce the search depth of the MCTS and reduce the computational complexity.

```
Procedure MCTS-Method (S_c,t,med,O,M,node,d)
1:  if d > d_max then return S_c,Q_d endif
2:  subtasks←med(S_c,t)
3:  if subtasks is empty then
4:    node.Q←0;node.N + 1;
5:    return S_c,node.Q
6:  endif
7:  for subtask in subtasks
8:    if subtask is a compound task then
9:      if subtask has been simulated then
10:       newnode←next node in node.children
11:     else newnode←Exps_TreeNode(t, 'task')
12:       add newnode into node.children endif
13:     S_c,index,Q_o←MCTS-Task(S_c,t,O,M,node,d+1)
14:   else if subtask is a primitive task then
15:     if subtask has been simulated then
16:       newnode←next node in node.children
17:     else newnode←Exps_TreeNode(t, 'operator')
18:       add newnode into node.children endif
19:     S_c,Q_t←MCTS-OperatorEvlt(S_c,t,O,M,node,d+1)
20:   endif
21: endfor
22: node.N + 1
23: node.Q←the average of the Q values of nodes in node.children
24: return S_c,node.Q
End MCTS-Method
```

The MCTS-OperatorEvlt function is used for simulating leaf nodes, leaf nodes are nodes utilized for evaluation in MCTS. Only when the leaf node is searched (or reach the maximum search depth), the final simulation results will be traced. The evaluation of the leaf node is represented by the CalValue function in the pseudo code. Different evaluation methods will have different effects on the final goal of the search tree.

```
Procedure MCTS-OperatorEvlt($S\_c$,$t$,$O$,$M$,$node$,$d$)
1:   $operator \leftarrow O(t)$; $S\_c \leftarrow operator(S\_c,t)$
2:   $Q\_o \leftarrow$ CalValue()
3: return $S\_c$, $Q\_o$
End MCTS-OperatorEvlt
```

In order to get the optimal solution of the planning, the objective function of the optimal problem can be defined in CalValue function. If the optimal goal of HTN planning is the length of the solution, the CalValue function can be calculated using the depth of the leaf node; if the optimal goal of HTN planning is the execution efficiency of the solution, the CalValue function can be computed according to the value defined by the action represented by the leaf node. For example, if the leaf node is defined as the execution time of an action, the optimal goal is the shortest time for planning solution execution; if the leaf node is defined as the performance value of the action, the optimal goal is the best performance of planning solution execution. If HTN is planned under the condition that the action execution effect is uncertain, the CalValue function can be calculated according to the execution success rate of the action represented by the leaf node. Multiple simulations based on the success rate can obtain a comprehensive optimal solution under uncertain conditions, which is the biggest advantage MCTS brings to HTN planning. The preponderance of applying MCTS to plan is that the CalValue function can contain heuristic search, so it can be compatible with the original distance heuristic search. Formula 1 represents the calculation method of the CalValue function, where ω_n refers to the weight, which can be set to comprehensively consider various optimization goals and simulate the final result.

$$Q_o = \omega_1 d + \omega_2 t + \omega_3 v + \omega_4 p + \omega_5 h + \cdots \omega_n x \tag{1}$$

Another important process in MCTS is "selection". In this study, the task node needs to select the next method node to simulate the search. This process is done through the Selection_Med function. The most classic "selection" method is accomplished by balancing the Q and N of the child nodes [29,31]:

$$m_n = argmax\left(q + c\sqrt{\ln(N)/n}\right) \tag{2}$$

where q and n are the value and the number of simulations of the child node respectively, and N represents the number of times the root node is simulated.

```
Procedure Selection_Med ($node$)
1:   $nodes \leftarrow node.children$
2:   $Q\_c \leftarrow$ NULL
3:   for $nod$ in $nodes$
4:      $q1 \leftarrow nod.Q$
5:      $q2 \leftarrow c *$ sqrt(ln($node.N$) /$node.n$[$nod.index$])
6:      $q \leftarrow q1 + q2$
7:      add $q$ into $Q\_c$
8:   endfor
9:   $index \leftarrow$ the index of the node on which the max value of $Q\_c$
10:  return $index$
End Selection_Med
```

Fig. 2 shows the four processes of MCTS in this study, where "Selection" is done through the Selection_Med function; "Expansion" is completed by Exps_TreeNode in each function. If the expanded node is a "task" or "method" node, it will continue to search down, if the expanded node is an "action" node, "Evaluation" will be performed; "Evaluation" is completed by performing a random simulation of the new extended node, when simulating to leaf nodes (or maximum search depth), the MCTS-OperatorEvlt function is used to evaluate the leaf nodes and provide the basis for "Backpropagation"; "Backpropagation" will transfer the evaluation value from the leaf node to the parent node through the iterative calculation of Q and N values in each function.

3.3. Shop-m algorithm combined MCTS-HTN

The SHOP algorithm decomposes tasks in a sequential manner, thus, the MCTS-HTN only needs to be executed before selecting the decomposition method, then sort the methods according to the Q or N value returned by the function. Finally, the MCTS-HTN can be combined with the SHOP algorithm by decomposing the task according to the new methods order, which is called SHOP-m algorithm, the pseudocode is as follows:

```
Procedure SHOP-m ($S$,$TN$,$O$,$M$)
1:   if $TN$=$NULL$ then return $NULL$ endif
2:   $t \leftarrow$ the first task in $TN$;$U \leftarrow$ the remaining task in $TN$
3:   if $t$ is a compound task then
4:      $S\_c \leftarrow S$
5:      $Q$,$N \leftarrow$ MCTS-HTN($S\_c$,$TN$,$O$,$M$)
6:      $Ms \leftarrow M(t)$;sort $Ms$ according to $Q$,$N$
7:      choose the first $m$ in $Ms$ for $t$
8:      $subtask \leftarrow$ decompose $t$ according to $m$
9:      return SHOP-m($S$,$subtask$+$U$,$O$,$M$)
10:  else if $t$ is a primitive task then
11:    $Op \leftarrow O(t)$;choose the first $o$ in $Op$ for $t$
12:    $S \leftarrow S(o)$
13:    return SHOP-m($S$,$U$,$O$,$M$)
14:  endif
End SHOP-m
```

If the problem to be planned has m tasks to be decomposed, each task has an average of n_1 methods, and the average number of subtasks by each method is n_2(includes tasks and operators). The total number of nodes in the search tree is N (the sum of the three types of nodes). In the case of n times of simulation, the time complexity of the algorithm is as follows:

$$O\left(m \cdot n \cdot n_2^{\frac{\lg N}{\lg(n_1+n_2)-\lg 2}}\right)$$

where the index represents the average search depth of the search tree when the number of nodes is N.

The original SHOP algorithm can only be applied to static planning. According to the idea of integrating planning and acting, SHOP-m can be extended to dynamic planning. In a dynamic environment, SHOP-m can plan the optimal solution under the current environment in real time according to changes in the external environment (or mission goals). The specific planning process is shown in Fig. 3.

According to Fig. 3, the basic idea of SHOP-m planning in a dynamic environment is: each time SHOP-m is called for planning, only a few of actions in the current state are planned (assuming there are N actions, which are illustrated by circles). Then these N actions are executed in order in the real environment (Represented by a red circle in Fig. 3). Before executing each action, it is necessary to judge whether the current environment still satisfies the execution of the action. If not, give up the next action, call SHOP-m to replan the next N actions. For example, after executing a_1 to a_i, if the current environment does not satisfy the execution premise of a_{i+1}, abandon the actions of a_{i+1} to a_n and replan to obtain the actions of a_{n+1} to a_{2n}. At this point, a_{n+1}

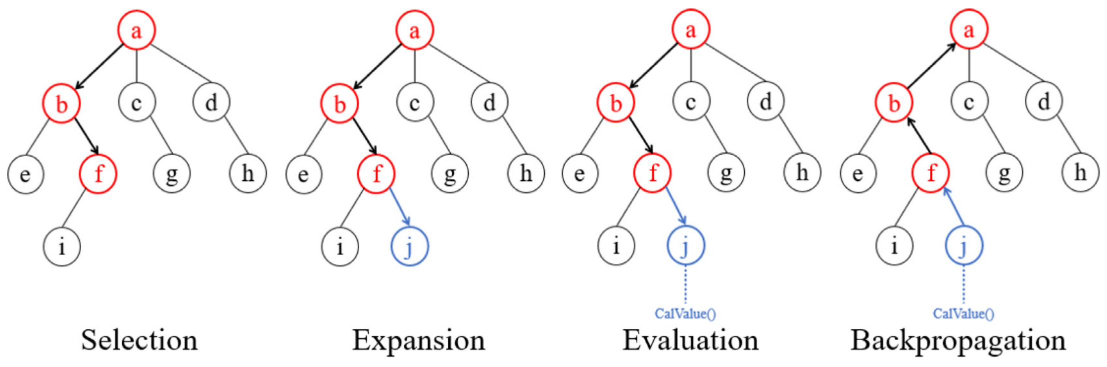

Fig. 2. The process of MCTS.

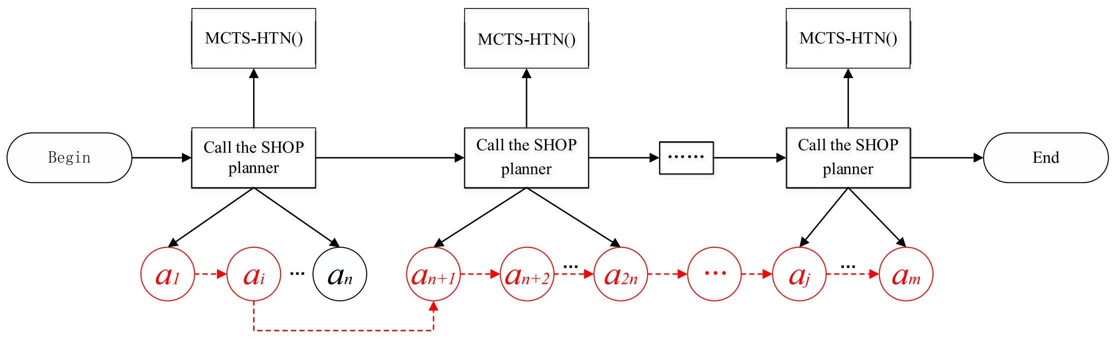

Fig. 3. Planning and executing process of SHOP-m in dynamic environment.

is the *i*+1th action to be executed in the overall execution. If all the actions obtained by a plan can be executed in the current environment, after *N* actions are executed, SHOP-m is called again to plan the next *N* actions. Repeat the above behavior, the plan can be completed in the dynamic environment. Furthermore, due to the MCTS-HTN, the best COA can always be obtained.

4. Experimental evaluation

According to the Python version of SHOP—Pyhop [32], SHOP-m is implemented in Python, which is called Python-m, and it is compared with Pyhop and Pyhop-h. A classical path planning problem is utilized the same as [15]: given a starting point and an ending point in an n×m grid and plan the optimal path between the two points, as shown in Fig. 4. We compare the performance of three algorithms in four experiments—Experiment one makes a preliminary comparison of three algorithms; Experiment two studies whether the barriers between the grids will affect the planning results; Experiment three researches if dynamic factors can have an influence on the planning results or not; Experiment four studies whether the uncertainty of action execution will have an effect on the planning results. We take three parameters of the starting and ending point locations, the grid size and the method order as independent variables, and the three parameters of the planning step size, the planning success rate and the optimal solution rate are taken as dependent variables. All the experiments run on a 2.80 GHz quad-core machine with 8 GB RAM.

The domain knowledge of this problem includes four methods and five operations. The four methods are described as follows.

$$m_1 = (Move, At\ (x, y)\ , [TurnNorth, MoveOneStep])$$

$$m_2 = (Move, At\ (x, y)\ , [TurnEast, MoveOneStep])$$
$$m_3 = (Move, At\ (x, y)\ , [TurnSouth, MoveOneStep])$$

$$m_4 = (Move, At\ (x, y)\ , [TurnWest, MoveOneStep])$$

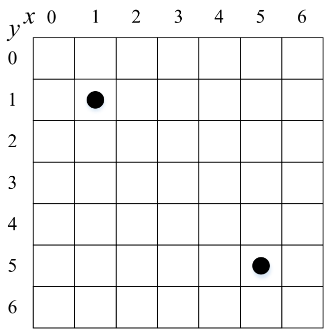

Fig. 4. Example of the path planning problem.

Every method is to solve the compound task of "*Move*". The preconditions for them are that the current target position is "*At* (*x*, *y*)". The four methods decompose the compound task into move one step to North, move one step to East, move one step to South, or move one step to West, respectively. All of them involve two subtasks: turning and moving one step, which are both primitive tasks.

The five operations in domain knowledge are *TurnNorth*, *TurnEast*, *TurnSouth*, *TurnWest* and *MoveOneStep*. They are described as follows.

$$opr_1 = (TurnNorth, At\ (x, y)\ , [face(North)])$$

$$opr_2 = (TurnEast, At\ (x, y)\ , [face(East)])$$
$$opr_3 = (TurnSouth, At\ (x, y)\ , [face(South)])$$

$$opr_4 = (TurnWest, At\ (x, y)\ , [face(West)])$$

$$opr_5 = (MoveOneStep, At\ (x, y)\ , [At(x', y')])$$

Where the first four operations are turning operations, which means turning the direction of the current point into the direction

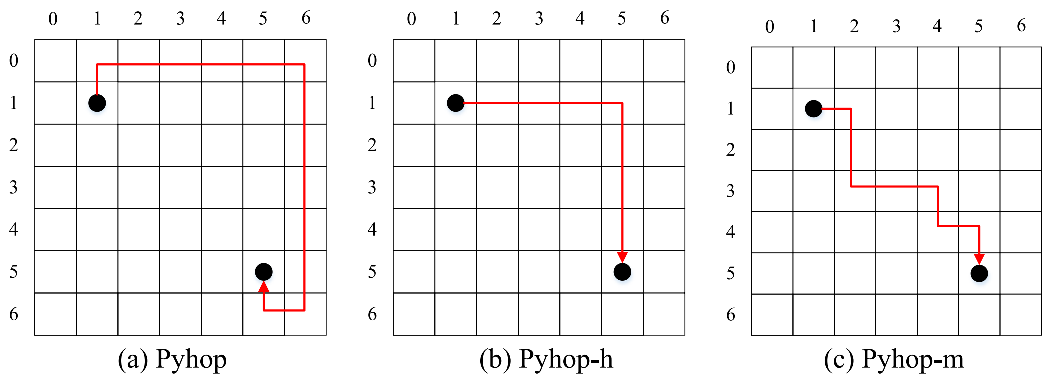

Fig. 5. Planning result in Experiment one.

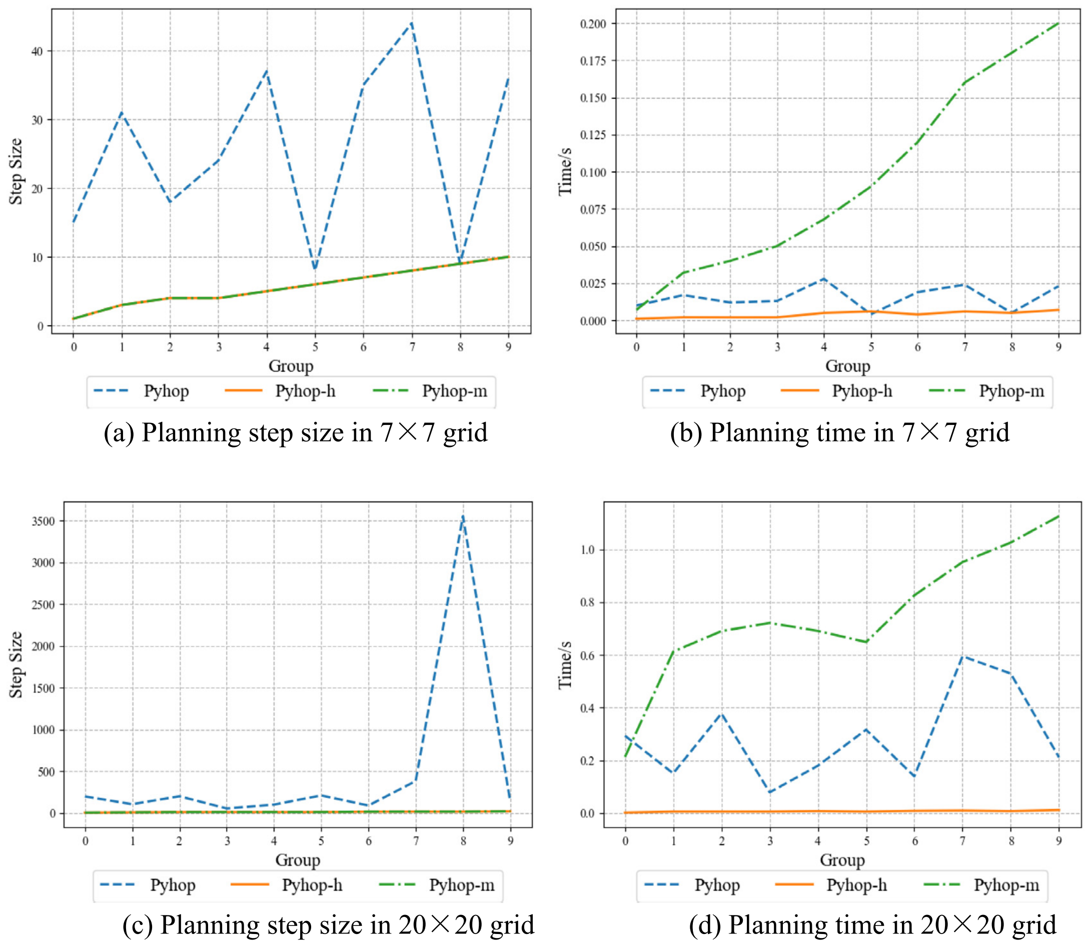

Fig. 6. Planning results in different planning spaces.

of the target. The fifth operation is the moving operation, which means moving forward one step in the current direction. For example, if the current direction is $face(North)$, the effect $At(x', y')$ in opr_5 is instantiated as $At(x, y - 1)$.

4.1. Experiment one

The Pyhop, Pyhop-h and Pyhop-m algorithms were compared in experiment one preliminarily. Because the optimal goal of path planning is the shortest step size, the heuristic function of distance combined with the depth d is used in the CalValue function to calculate the estimation values of leaf nodes. The number of simulation times is 200 and the maximum search depth is 100. It should be emphasized that we find it more reliable to use the array N to guide HTN selection decomposition method than array Q through experiments. In other words, the more times a decomposition method is simulated, the better it is. It is the same as the way of selecting the next drop point in AlphaGo, which indicates that our algorithm has high credibility.

In the 7×7 grid, the position of initial point is (1,1), and the position of target point is (5,5). The order of decomposition methods is med_1, med_2, med_3, med_4. Fig. 5 reveals the planning results of the three algorithms.

According to Fig. 5, the step size of the solution obtained by Pyhop planning is up to 14 steps, as shown in Fig. 5(a). On the other hand, although the paths of Pyhop-h and Pyhop-m are

Table 1
Planning results with different order of decomposition methods.

Initial	Target	Order1	Order2	Order3	Order4	Pyhop-m/h
4,5	5,5	15	1	15	13	1
3,3	2,1	31	∞	39	35	3
5,0	3,2	18	12	∞	38	4
4,0	3,3	24	20	∞	32	4
1,2	2,6	37	29	5	7	5
0,1	6,1	8	6	16	36	6
1,0	2,6	35	43	7	9	7
5,1	0,4	44	22	46	8	8
2,5	6,0	9	∞	11	41	9
6,0	1,5	36	36	36	24	10

different, both algorithms only need 8 steps, which is the shortest distance between two points, as shown in Figs. 5(b) and 5(c). The results show that compared with the pure Pyhop, Pyhop-m and Pyhop-h can both plan the shortest path under this condition.

In order to illustrate the influence of the size of planning space on the planning, we randomly generated 10 groups of initial points and target points in a 7×7 grid and a 20×20 grid, and three algorithms are used to calculate the planning time and step of size, respectively. The results are shown in Fig. 6. The blue dotted line represents the Pyhop algorithm, the orange solid line represents the Pyhop-h algorithm, and the green dotted line represents the Pyhop-m algorithm.

The results in Fig. 6 show that Pyhop-m and Pyhop-h can both plan out the shortest path regardless of the size of the planning space, and the planning step size of Pyhop is always larger than the previous two. Pyhop can even go to extremes, such as the 9th point in Fig. 6(c). However, the planning time of Pyhop-m is longer than Pyhop-h, which is expected, because the simulation brings more time consumption.

In order to illustrate the influence of the written order of decomposition methods on the planning, we randomly select four groups of decomposition methods in different order. We use 10 groups of initial points and target points randomly generated in a 7 × 7 grid. Three algorithms are used to calculate the planning step size. The results are shown in Table 1, where Order1, Order2, Order3 and Order4 respectively represent the decomposition methods written in the order of $\langle m_1, m_2, m_3, m_4\rangle$, $\langle m_2, m_4, m_3, m_1\rangle$, $\langle m_3, m_2, m_1, m_4\rangle$, $\langle m_4, m_3, m_2, m_1\rangle$. Pyhop-m/h represents the step size calculated by Pyhop-m and Pyhop-h, and the symbol ∞ in the table represents no planning solution. To reveal the results more intuitively, two groups are selected to draw histograms as shown in Fig. 7.

As can be seen from Table 1 and Fig. 7, for each group of experiment points, Order1 to Order4 generally have different plan length, some of which are the best ones, while others cannot even plan out the results (∞). This demonstrates that the difference of writing sequences has a great impact on the planning results of Pyhop algorithms. For the same problem, some sequences have good capability in solving the problems, while others require long steps or even cannot solve the problem. In contrast, Pyhop-m and Pyhop-h retain the original and optimal planned results. The results indicate that Pyhop-m and Pyhop-h can separate the planning solutions from the decomposition methods, thus reducing the burden on domain experts to write domain knowledge.

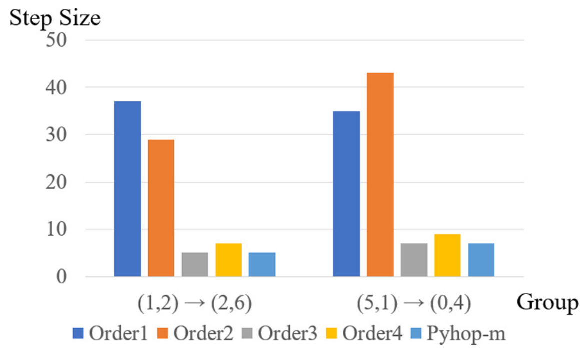

Fig. 7. Two planning results with different order of decomposition methods.

4.2. Experiment two

In experiment one, because the planning problem is relatively simple, the Pyhop-m has the same performance as the Pyhop-h. Therefore, we expand the space of the planning problem to a grid with obstacles, so as to study whether the obstacles between the grids will affect the planning results.

First of all, we provide an extreme method of setting obstacles to illustrate the problem, as shown in Fig. 8(a). The red and black frame indicates that there is a one-way barrier between adjacent grids, and the red arrow means that two adjacent grids cannot pass through. In this case, Pyhop-h and Pyhop-m are utilized to make a second plan. The planning results are shown in Figs. 8(b) and 8(c).

The results in Figs. 8(b) and 8(c) illustrate that the step size of solution obtained by Pyhop-h is 12, while the step size of solution obtained by Pyhop-m is 10. The difference between the two is that a greedy strategy is adopted in Pyhop-h and it only considers the best “next step”. Whereas, the Pyhop-m takes the global planning into account through simulation, so as to get a more optimized solution.

For a more general illustration, we use 10 groups of randomly generated initial and target points in the 7×7 grid in experiment one. For each set of points, 30 obstacles are randomly generated. When planning in the case of obstacles, each group of initial and target points are planned 100 times (30 different obstacles are generated each time), and the average step size of 100 times is obtained through planning. The results are shown in Table 2, where Step_h and Step_m mean the confidence interval of step size by using pyhop-h algorithm and Pyhop-m algorithm with the confidence level of 95%, respectively. p_1 represents the probability of planning success and p_2 represents the probability of obtaining the optimal planning solution in the successful planning, i.e., the rate of the problems whose solutions are optimal to the problems that are successfully planned. For example, if 100 problems are planned, and 90 problems return the planning solutions, 80 problems return the optimal solution, then the value of p_1 is 90%, and the value of p_2 is 88.89%. P$_1$_h and P$_1$_m represent p_1 values of Pyhop-h and Pyhop-m, respectively, and P$_2$_h and P$_2$_m represent p_2 values of Pyhop-h and Pyhop-m, respectively. In order to make it more intuitively, the average step size, the average planning success rate and the average optimal solution rate of the 10 groups of points are represented in a histogram as shown in Fig. 9, where the blue and green bars respectively represent the average results obtained by Pyhop-h algorithm and Pyhop-m algorithm.

As can be seen from Table 2, for each set of experimental points, the upper and lower bounds of the step size interval obtained by Pyhop-m algorithm are all smaller than those obtained by Pyhop-h algorithm, and the interval range of the former is

Table 2
Comparison of planning results with obstacles.

Initial	Target	Step_h	Step_m	P_1_h	P_1_m	P_2_h	P_2_m
1,0	2,6	[8.29,9.27]	[7.77,8.34]	81.00%	95.00%	72.83%	98.94%
3,3	2,1	[3.33,4.53]	[3.07,3.86]	97.00%	99.00%	87.63%	97.98%
4,5	5,5	[1.27,2.06]	[1.22,1.67]	99.00%	99.00%	96.97%	98.99%
5,0	3,2	[4.26,4.76]	[4.17,4.55]	74.00%	100.00%	86.49%	100.00%
6,0	1,5	[10.88,11.69]	[10.08,10.38]	81.00%	95.00%	60.49%	96.84%
1,2	2,6	[6.33,7.72]	[5.28,5.66]	88.00%	98.00%	69.32%	100.00%
2,5	6,0	[9.65,10.98]	[9.07,9.49]	73.00%	94.00%	68.49%	97.87%
5,1	0,4	[9.26,10.35]	[8.23,8.68]	83.00%	97.00%	63.86%	98.97%
0,1	6,1	[7.30,8.08]	[7.29,7.86]	77.00%	90.00%	90.91%	100.00%
4,0	3,3	[4.63,5.86]	[4.28,4.77]	90.00%	99.00%	87.78%	97.98%

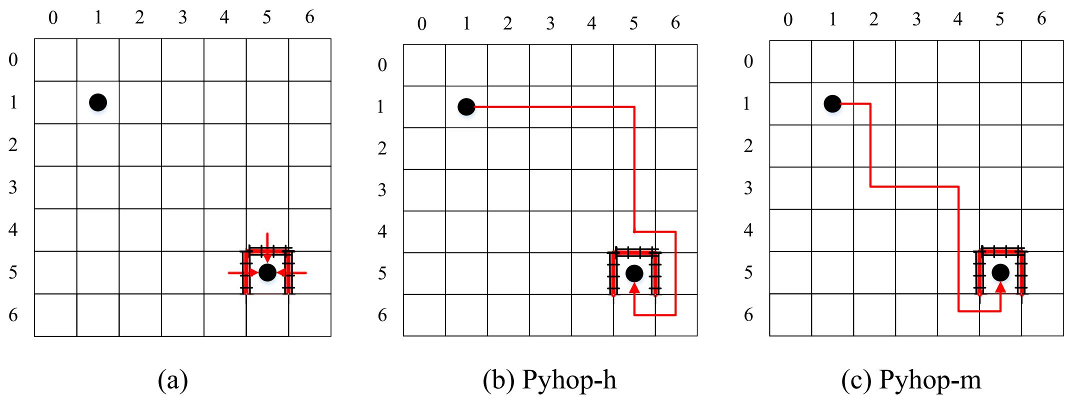

Fig. 8. Planning results with obstacles.

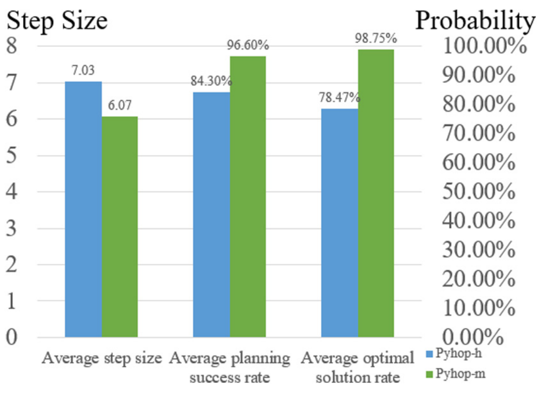

Fig. 9. Average results in Experiment one.

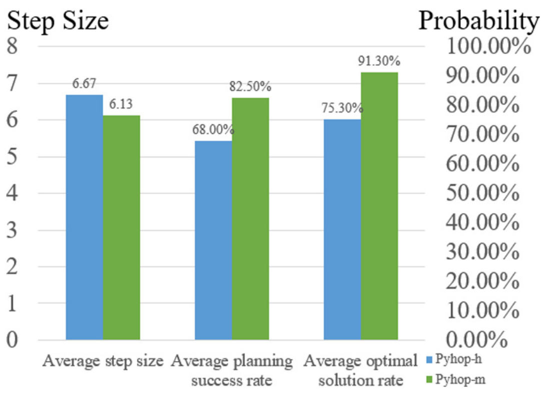

Fig. 10. Average results in Experiment three.

smaller than that of the latter, which indicate that the Pyhop-m algorithm can obtain smaller plan length with higher stability. By calculating the confidence intervals of four probabilities at 10 points, the confidence intervals of the planning success rate and the optimal solution rate of the Pyhop-m algorithm are [94.78%,98.42%] and [98.12%,99.39%], respectively, while the confidence intervals of the Pyhop-h algorithm are [78.99%,89.61%] and [70.93%,86.03%], which reveal that the success rate and the optimal rate of the former algorithm are higher and more stable. In terms of average step size, average planning success rate and average optimal rate of the 10 experimental points shown in Fig. 9, the Pyhop-m algorithm is superior to Pyhop-h algorithm in 13.7%, 12.3% and 20.28%, respectively. The experimental results show that Pyhop-m has better adaptability and planning performance for path planning with obstacles.

4.3. Experiment three

Experiment one and two illustrate that Pyhop-m is better than Pyhop-h in planning step size, planning success rate and optimal solution rate of planning results in static environment. Nonetheless, in practice, the external environment is always dynamic. The change of the environment will result in the premise of the primitive task is no longer satisfied, thus affecting the execution

Table 3
Planning result in dynamic environment.

Initial	Target	Step_h	Step_m	P_1_h	P_1_m	P_2_h	P_2_m
1,0	2,6	[7.93,8.66]	[7.45,7.90]	59.00%	83.00%	73.49%	82.61%
3,3	2,1	[3.34,4.13]	[3.21,3.50]	80.00%	85.00%	77.23%	88.17%
4,5	5,5	[1.11,1.69]	[1.22,1.48]	94.00%	98.00%	89.71%	95.86%
5,0	3,2	[4.27,4.66]	[4.23,4.55]	70.00%	96.00%	79.76%	93.88%
6,0	1,5	[11.34,12.00]	[10.27,10.68]	70.00%	82.00%	59.30%	94.38%
1,2	2,6	[5.73,6.35]	[5.21,5.52]	63.00%	79.00%	71.79%	94.38%
2,5	6,0	[9.57,10.67]	[9.06,9.29]	29.00%	57.00%	71.01%	92.41%
5,1	0,4	[9.16,9.73]	[8.02,8.66]	70.00%	75.00%	67.90%	95.45%
0,1	6,1	[6.61,7.06]	[6.61,6.97]	74.00%	85.00%	74.16%	85.82%
4,0	3,3	[4.23,5.17]	[4.26,4.62]	71.00%	85.00%	88.66%	90.11%

of COA. Under these circumstances, it is necessary to replan and obtain the new COA to complete the remaining tasks in the current environment.

The HTN planning algorithm has the ability of coping with the impact of the dynamic environment by planning the remaining tasks in the current environment, that is, by taking the current unfinished task and the current environment as the initial task network and the initial environment, it restarts the planner and gets the new COA. To test the planning ability of Pyhop-m under the dynamic environment, add certain randomness to the obstacles on the basis of experiment two. At the beginning of the planning, there is no obstacle in grid, and at certain intervals, a set number of obstacles will appear or change randomly in the grids. The total number of obstacles in the grids is certain, i.e., the emergence of a new obstacle must be accompanied by the disappearance of an old obstacle.

We still take the 7×7 grid as an example. 10 groups of initial and target points randomly generated in experiment one are used. For each group of points, 30 obstacles will be generated dynamically and repeatedly during the execution of actions, thus triggering the replanning of Pyhop-h or Pyhop-m until the final task is completed or the planning fails. In order to reduce the search depth and facilitate the search efficiency, Pyhop-m only plans five steps at each time. Each group of points is still planned for 100 times, and the experimental results are shown in Table 3 and Fig. The symbols in Table 3 and Fig. 10 have the same meanings as in Table 2 and Fig. 9.

As can be seen from Table 3, the experimental results under dynamic conditions are similar to Experiment 2. For each set of the experimental points, the Pyhop-m algorithm is able to obtain smaller plan length with higher stability. By calculating the confidence intervals of four probabilities at 10 points, the confidence intervals of the planning success rate and the optimal solution rate of the Pyhop-m algorithm are [75.82%,89.18%] and [88.67%,93.95%], respectively, while the confidence intervals of the Pyhop-h algorithm are [58.19%,77.81%] and [69.91%,80.69%], which indicate that the former algorithm also has a higher and more stable success rate and the optimal rate in the dynamic environment. In terms of average step size, average planning success rate and average optimal rate of the 10 experimental points shown in Fig. 10, the Pyhop-m algorithm is superior to Pyhop-h algorithm in 8.1%, 14.5% and 16%, respectively. By comparing Figs. 9 and 10, it can be found that under dynamic environment, the average step size of Pyhop-h is shorter, this is because the local greed is optimized through multiple planning, and the average step size of Pyhop-m using the global optimal algorithm change little. On the other hand, the uncertainties brought by dynamic environment decrease the planning success rate and optimal solution rate of the two algorithms, and lead to planning failure or a "dead end". But overall, Pyhop-m handles dynamic factors better than Pyhop-h.

4.4. Experiment four

The biggest advantage of the Monte Carlo method is the simulation, so our algorithm obtains a good solution of uncertain action execution problems. In practical application, the result of action execution may be uncertain [11], i.e., the environmental impact brought by action is not completely knowable, but fluctuates within a certain range. For example, the artillery fire landing site in military is not fixed but within a certain range. Most HTN planning algorithm can only solve the problems with certain action executions. For the case of uncertainty, they can only randomly select to execute one of these effects. However, for Pyhop-m, multiple simulations can be performed on leaf nodes, and one execution effect can be selected according to rules for each simulation. The simulation results of multiple times can be integrated as the return value to upper nodes for reference.

So as to verify the planning performance of Pyhop-m in uncertainty problems, on the basis of experiment two, experiment four add a "pass rate" attribute to every obstacle, that is, each obstacle is not completely impassable, but there are five grades of 0%, 20%, 40%, 60% and 80%. For the 20% grade, it means that there is a 20% chance that the current grid can be passed to the target grid. As a result, the effect of executing the *MoveOneStep* action is not certain, and it depends on different probabilities to decide whether to across to the next grid.

Experiment four is based on the static planning of experiment two, and 30 random obstacles are set with different "pass rate" (the pass rate in experiment two are all 0%). We still use 10 randomly generated initial and target points in the 7×7 grid of experiment one. For each group of points, we randomly generated 30 obstacles, each obstacle is randomly set to one of the five grades. For Pyhop-h, if the "pass rate" is not 0%, it is assumed that the grid can be crossed directly, and this action will be added to the COA. Each group of initial and target points is planned 100 times (each time generating 30 different obstacles). Each COA execute 100 times in practice. If a certain action needs to pass through a grid with obstacles, it will select between "across" and "not across" according to the "pass rate". If "not across" is selected, the step size will be increased by 1, and the action will continue to execute until the target point is reached. The total step size of each execution is recorded, and the average step size of 100 executions is calculated as the step size of this planning.

Table 4
Planning results in uncertainty problems.

Initial	Target	Step_h	Step_m	Step_m$_w$
1,0	2,6	[8.47,9.41]	[8.37,9.30]	[7.43,7.98]
3,3	2,1	[3.79,4.37]	[3.44,3.99]	[3.03,3.16]
4,5	5,5	[1.22,1.66]	[1.22,1.63]	[1.21,1.61]
5,0	3,2	[4.70,5.29]	[4.51,5.10]	[4.06,4.19]
6,0	1,5	[11.96,12.90]	[11.94,12.91]	[10.03,10.09]
1,2	2,6	[6.02,6.98]	[5.81,6.45]	[5.16,5.47]
2,5	6,0	[10.63,11.64]	[10.23,11.04]	[9.04,9.16]
5,1	0,4	[9.59,10.63]	[9.15,10.00]	[8.10,8.41]
0,1	6,1	[7.37,8.35]	[7.27,8.13]	[6.99,7.52]
4,0	3,3	[4.78,5.66]	[4.84,5.52]	[4.13,4.43]

Finally, the average step size of 100 planning is computed as the step size result of this pair of experimental points.

In order to explain the influence of different evaluation functions of leaf nodes on Pyhop-m planning, the evaluation function in experiment two is expanded and "pass rate" is taken as one of the parameters of the evaluation function, as shown in formula (3). The experimental results are shown in Table 4, Where Step_h represents the confidence interval of the plan length obtained by Pyhop-h algorithm, Step_m denotes the confidence interval of the plan length obtained by using the leaf node evaluation function of Experiment 2, and Step_m$_w$ represents the confidence interval of the plan length obtained by utilizing formula (3) as the evaluation function. The confidence levels are all 95%.

$$Q_o = \omega_1 d + \omega_2 h + \omega_3 p \tag{3}$$

As can be seen from the results in Table 4, the upper and lower bounds of the confidence intervals of Step_h, Step_m and Step_m$_w$ gradually decrease, and the interval range also getting smaller. These show that in the problem with uncertain execution effects, the HTN planning algorithm based on MCTS is superior to that based on heuristic function, whether the evaluation function of leaf node is considered or not. For the result of Step_m$_w$ is better than Step_m, this is because the evaluation function used by Step_m$_w$ takes into account the influence of actions with uncertain execution effect when evaluating leaf nodes. However, within a certain computing capacity, the more appropriate evaluation function is, the better planning result will be. Hence, the selection of appropriate evaluation functions according to different planning objectives is an important factor for using MCTS to promote the planning performance of HTN.

From experiment one to experiment four, it can be shown that Pyhop-m is superior to Pyhop and Pyhop-h in plan step size, planning success rate and optimal solution rate. In addition, Pyhop-m has stronger planning ability to adapt to dynamic environment, and in some cases, Pyhop-m can resolve problems that Pyhop and Pyhop-h cannot solve, such as the problems with uncertain action executions.

5. Conclusions and future work

In this paper, we use MCTS to guide how to choose the best decomposition method for compound tasks in HTN planning, thereby improving the planning performance of HTN. We first expand the formal description of HTN and add "planning node" to link MCTS and HTN planning. Then MCTS-HTN is proposed as the entry function for calling MCTS in HTN planning, and the calculation formula of CalValue function is given for MCTS leaf node evaluation. The leaf node evaluation function can be modified according to different planning goals, so that the optimal solutions of planning can be obtained in different planning problems. Finally, we combine MCTS with the classic planning algorithm, SHOP, and establish a new HTN planning algorithm, SHOP-m, which is implemented in Python, therefore we call it Pyhop-m. Pyhop-m has the ability of planning tasks in both static and dynamic environments. In addition, it makes HTN planning insensitive to the input order of the methods and reduces the writing burden of domain experts. Furthermore, it can converge to the optimal solution of the planning from the perspective of global optimization under the condition of unlimited computing power. Experimental results show that whether in a static or dynamic environment, Pyhop-m is superior to Pyhop algorithm and the heuristic function-based Pyhop-h algorithm in planning step size, planning success rate and optimal solution rate. Moreover, Pyhop-m can solve the planning problems with uncertain action execution by means of multiple simulations and evaluations of MCTS leaf nodes, which is impossible for Pyhop and Pyhop-h algorithms. Although the adopted experimental framework is relatively simple, the planning ideas for most optimization problems can be met by the way of constructing the objective function of optimization problem on the basis of CalValue function and using MCTS to solve the problem. In future work, the value of this algorithm in practical industrial problems such as network routing service and material selection for manufacturing can be studied.

Although Pyhop-m has achieved good results in the experiment, there are still some problems that need to be resolved in the future research. First of all, our method is completely feasible when the external environment is absolutely observable, but when the external environment is partially observable, there may be inconsistencies in planning and execution, which requires to find a unified mapping method from complex execution model to a simple planning model. "The external environment is absolutely observable" means that the planner knows all the state information in the planning space during the planning. Even if the external environment changes, the planner can timely know the changed part. For example, the planning space of Go is completely observable. "The external environment is partially observable" means that the planner is not able to observe some information in the planning space during planning. For example, the planning space of MOBA games such as Dota is partially observable. The Hierarchical Domain Definition Language was expanded in [33], which enhanced the connection between various problem fields. It can be considered to combine this language with MCTS to better solve the inconsistencies in planning and execution. Secondly, although the usage of MCTS to guide HTN planning has the advantage in planning results, as explained in [34], the use of search trees in HTN comes with large resource occupation. The planning time of Pyhop-m in Experiment one is significantly longer than that of Pyhop-h. Similarly, the optimal rate of planning solution of Pyhop-m in Experiment two does not reach 100%, which are both caused by insufficient planning resources and insufficient tree search convergence. In terms of complexity, although the time complexity of the algorithm is exponential, when the scale of the problem is fixed, the complexity is only related to the number of simulation times n and the average number of methods n_1 and the average number of subtasks n_2 are not large. Therefore, the exponential will not have the big influence when the problem size is not very large. However, when the scale of the problem is large, the running time of the algorithm will increase rapidly with the increase of the tree depth. If the average number of subtasks of the method is kept unchanged (n_2), and the average number of methods of the task is increased (n_1), the total time complexity will be reduced, but this results in a decrease in the accuracy of the simulation. Hence, how to perform effective depth pruning and breadth pruning of the search tree in planning is the focus of future research on this method and the key to the application of this method in actual planning projects. Finally, in Experiment three of this paper, the planning ability of Pyhop-m in dynamic environment is stronger than Pyhop-h, mainly due to its strong ability in static planning. However, by using the planning tree produced in the MCTS process, Pyhop-m can guide the next replanning. Therefore, in the dynamic environment, how does MCTS intervene in replanning, when to replan, and whether it begins with repairation or replanning after intervention [10], how to utilize the planning tree of the last plan during replanning, they are all the hot spots for future research on this method.

CRediT authorship contribution statement

Tianhao Shao: Conceptualization, Methodology, Writing - original draft. **Hongjun Zhang:** Supervision. **Kai Cheng:** Data curation, Software. **Ke Zhang:** Writing - review & editing. **Lin Bie:** Validation.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

We thank Dana Nau for implementing Pyhop and providing open download. This work was supported by the National Natural Science Foundation of China (No. 61806221).

References

[1] H.W. Wang, C. Qi, Hierarchical Task Network Planning Based Emergency Response Decision Making Theory and Method, Science Press, Beijing, 2015.
[2] G. Robertson, I. Watson, A review of real-time strategy game AI, AI Mag. 35 (4) (2014) 75–104.
[3] T. Gateau, C. Lesire, M. Barbier, Hidden: Cooperative plan execution and repair for heterogeneous robots in dynamic environments, in: Proceedings of the 2013 IEEE/RSJ International Conference on Intelligent Robots and Systems, 2013, pp. 4790–4795.
[4] A. Hristoskova, B. Volckaert, F.D. Turck, Dynamic composition of semantically annotated web services through QoS-aware HTN planning algorithms, in: Proceedings of the 2009 Fourth International Conference on Internet and Web Applications and Services, 2009, pp. 337–382.
[5] I. Georgievski, M. Aiello, HTN planning: overview, comparison, and beyond, Artificial Intelligence 222 (2015) 124–156.
[6] E.D. Sacerdoti, The nonlinear nature of plans, in: Proceedings of the Fourth International Joint Conference on Artificial Intelligence, 1975, pp. 206–214.
[7] D. Nau, T.-C. Au, O. Ilgham, U. Kuter, J. William Murdock, D. Wu, F. Yaman, SHOP2: an HTN planning system, J. Artificial Intelligence Res. 20 (December) (2003) 379–404.
[8] D. Nau, T.-C. Au, O. Ilgham, U. Kuter, H. Muñoz Avila, J. William Murdock, D. Wu, F. Yaman, Applications of SHOP and SHOP2, Intell. Syst. IEEE 20 (2) (2005) 34–41.
[9] O. Ilghami, Documentation for JSHOP2, University of Maryland, 2006.
[10] T.H. Shao, H.J. Zhang, K. Cheng, C.Y. Dai, X.H. Yu, K. Zhang, A review of replanning in hierarchical task network, Syst. Eng. Electron. 495 (12) (2020) 171–184.
[11] H.W. Wang, D. Liu, P. Zhao, C. Qi, X. Chen, Review on hierarchical task network planning under uncertainty, Acta Automat. Sinica 42 (5) (2016) 655–667.
[12] A. Gerevini, U. Kute, D. Nau, A. Saetti, N. Waisbrot, Combining domain-independent planning and HTN planning: the duet planner, in: Proceedings of the European Conference on Artificial Intelligence, 2008, pp. 573–577.
[13] A. Gerevini, A. Saetti, I. Serina, Planning through stochastic local search and temporal action graphs, J. Artificial Intelligence Res. 20 (December) (2003) 239–290.
[14] N. Waisbrot, U. Kuter, T. Könik, Combining heuristic search with hierarchical task-network planning: a preliminary report, in: Proceedings of the International Florida Artificial Intelligence Research Society Conference, 2008, pp. 577–578.
[15] K. Cheng, L. Wu, X.H. Yu, C.X. Yin, R.Z. Kang, Improving hierarchical task network planning performance by the use of domain-independent heuristic search, Knowl.-Based Syst. 142 (Februrary) (2018) 117–126.
[16] R. Krogt, M. Weerdt, Plan repair as an extension of planning, in: Proceedings of the Fifteenth International Conference on Automated Planning & Scheduling, 2005, pp. 161–170.
[17] M. Elkawkagy, P. Bercher, B. Schattenberg, S. Biundo, Improving hierarchical planning performance by the use of landmarks, in: Proceedings of the Twenty-Sixth National Conference on Artificial Intelligence, AAAI 2012, 2012, pp. 1763–1769.
[18] M. Elkawkagy, B. Schattenberg, S. Biundo, Landmarks in hierarchical planning, in: Proceedings of the Twentieth European Conference on Artificial Intelligence, 2010, pp. 229–234.
[19] M.L. Li, H.W. Wang, C. Qi, A novel HTN planning approach for handling disruption during plan execution, Appl. Intell. 46 (4) (2016) 800–809.
[20] G. Behnke, D. Holler, S. Biundo, Finding optimal solutions in HTN planning – A SAT-based approach, in: Proceedings of the Twenty-Eight International Joint Conference on Artificial Intelligence, 2019, pp. 5500–5508.
[21] D. Holler, P. Bercher, G. Behnke, S. Biundo, On guiding search in HTN planning with classical planning heuristics, in: Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, 2019, pp. 6171–6175.
[22] D. Holler, P. Bercher, G. Behnke, S. Biundo, HTN planning as heuristic progression search, J. Artificial Intelligence Res. 67 (April) (2020) 835–880.
[23] S. Patra, M. Ghallab, D. Nau, P. Traverso, Acting and planning using opearational models, in: Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence, 2019, pp. 7691–7698.
[24] S. Patra, M. Ghallab, D. Nau, P. Traverso, Interleaving acting and planning using operational models, in: Proceedings of the International Conference on Automated Planning and Scheduling, 2019, pp. 46–54.
[25] S. Patra, M. Ghallab, D. Nau, P. Traverso, APE: An acting and planning engine, Cogn. Syst. 7 (December) (2019) 175–194.
[26] D. Nau, M. Ghallab, P. Traverso, Blended planning and acting preliminary approach, research challenges, in: Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, 2015, pp. 4047–4051.
[27] X. Xu, M. Yang, G. Li, Adaptive CGF commander behavior modeling through HTN guided Monte Carlo Tree Search, J. Syst. Sci. Syst. Eng. 27 (2) (2018) 231–249.
[28] S. Patra, J. Mason, A. Kummar, M. Ghallab, P. Traverso, D. Nau, Integrating acting, planning, and learning in hierarchical operational models, in: Proceedings of the 2020 International Conference on Automated Planning and Scheduling, 2020, ArXiv preprint arXiv:2003.03932v1.
[29] D. Silver, A. Huang, C.J. Maddison, Mastering the game of Go with deep neural networks and tree search, Nature 529 (7587) (2016) 484–489.
[30] D. Silver, J. Schrittwieser, K. Simonyan, Mastering the game of Go without human knowledge, Nature 550 (7676) (2017) 354–359.
[31] L. Kocsis, C. Szepesvari, Bandit based Monte Carlo planning, in: Proceedings of the 2006 European Conference on Machine Learning, 2006, pp. 282–293.
[32] Pyhop, 2020, Online: accessed Jun. http://bitbucket.org/dananau/pyhop.
[33] D. Holler, G. Behnke, P. Bercher, S. Biundo, H. Fiorino, D. Pellier, R. Alford, HDDL: An extension to PDDL for Expressing hierarchical planning problems, in: Procceding of Thirty-Forth AAAI Conference on Artificial Intelligence, 2020.
[34] R. Alford, Search complexities for HTN planning, Ki-Künstliche Inte. 30 (1) (2016) 99–100.