applied sciences

MDPI

Article

Learning and Planning Based on Merged Experience from Multiple Situations for a Service Robot

Zhixian Chen [1,2], Baoliang Zhao [1,2], Shijia Zhao [1,2], Ying Hu [1,2,*] and Jianwei Zhang [3,*]

1 Shenzhen Key Laboratory of Minimally Invasive Surgical Robotics and System, Shenzhen Institutes of Advanced Technology, Chinese Academy of Sciences, Shenzhen 518055, China; zx.chen@siat.ac.cn (Z.C.); bl.zhao@siat.ac.cn (B.Z.); sj.zhao1@siat.ac.cn (S.Z.)

2 Shenzhen College of Advanced Technology, University of Chinese Academy of Sciences, Shenzhen 518055, China

3 Technical Aspects of Multimodal Systems (TAMS), University of Hamburg, 22527 Hamburg, Germany

* Correspondence: ying.hu@siat.ac.cn (Y.H.); zhang@informatik.uni-hamburg.de (J.Z.); Tel.: +86-0755-8639-2182 (Y.H.)

Received: 25 August 2018; Accepted: 1 October 2018; Published: 6 October 2018

Abstract: For a service robot, learning appropriate behaviours to acquire task knowledge and deliberation in various situations is essential, but the existing methods do not support merging the plan-based activity experiences from multiple situations in the same task. In this paper, an abstract method is introduced to integrate the empirical activity schemas of multiple situations, and a novel algorithm is presented to learn activity schema with abstract methods. Furthermore, a novel planner called the schema-based optimized planner (SBOP) is developed based on the learned activity schema, in which actions merging optimization and partially backtracking techniques are adopted. A simulation with a PR2 robot and a physical experiment is conducted to validate the proposed method. The results show that the robot can generate a plan to recover from failure automatically using the novel learning and planning method, given that the experienced exception has been merged in the activity schema. Owing to the improved autonomy, the proposed SBOP exhibits increased efficiency in dealing with tasks containing loops and multiple activity schema instances. This research presents a novel solution of how to merge activity experiences from multiple situations and generate an intelligent and efficient plan that could adapt to a dynamically changing environment.

Keywords: task knowledge; activity schema; HTN planning; experiences merging; abstract method

1. Introduction

Today, an increasing number of robots are developed for service in daily human life, and these robots are expected to be more flexible and efficient. Moreover, robots working in the real world need to adapt to the dynamically changing human-inhabited environments. Thus, approaches to learning the appropriate behavior to acquire task knowledge and deliberation in various situations are essentially required [1]. Early work attempted to address high-level task learning from human demonstration based on experience-based planning domains (EBPDs) [2]. The task models refer to activity schemas composed of action sequences or operation plans and the features of objects involved in the operators. Furthermore, the activity schemas are used for grounding task vocabulary which is transmitted to the robot by the human user, and as a guide to increase the speed of plan searching. However, the approach attempted to abstract activity schema directly from the operators, without a middle layer to learn different action sequences for certain task classes, so it does not support merging the plan-based activity experiences from multiple situations in the same task.

Numerous studies have focused on obtaining robot activity experiences and conceptualizing the experiences in Web Ontology Language (OWL) ontology [3–5]. However, the problem of dynamically

storing acquired concepts in the robot's ontology has not been addressed to date. The problem of reinterpreting past episodes in terms of a new ontology [4,6] is a research topic under investigation by several groups; however, a perfect solution has not been developed to date. Generalization is the first step in conceptualizing a plan-based robot activity experience. Seabra presents several deductive and inductive transformation methods for the principles that explain the success of a failure-recovery strategy from experienced episodes [7,8]. Recently, conceptualizing activity experiences in the form of activity schemata that can be used to guide a planner towards a solution has attracted much attention from academia [7,9]. Following the idea, Vahid proposed a conceptualization approach based on deductive generalization, abstraction and feature extraction [10]. Then, a goal inference procedure is added to activity schemas, and the inferred goal is useful in solving different instances of the same problems [2]. In one study [11], the approach is extended to robot experiences containing loops. However, the learned activity schemas are fixed and separated from each other, which causes the difficulty of switching to activity schema when the current learned activity schema fails.

Hierarchical task network (HTN) planning is an approach that is adopted extensively for automated planning. Numerous studies focus on learning the applicability conditions from given structural traces of HTNs. Early studies, such as those utilizing the CaMeL algorithm [12] and DInCAD algorithm [13], assumed the complete observability of the intermediate states of the plans and a complete action model was given. Approaches such as the Icarus algorithm [14] and the HTN-MAKER algorithm [15] have been developed to learn the decomposition methods for HTN. It is difficult to apply such approaches because the observed plan traces lack complete and consistent information in real world situations. Hankz presented HTN-learner to learn the HTN method preconditions in HTN planning given a set of partially observed decomposition trees [16]. This method is inefficient for scenarios with complex tasks and uncertain environments, in which the number of predicates needed to be identified is large.

A plan solution guided by the activity schema is an effective means that uses the learned task knowledge. Schema-Based Planner (SBP), an adapted A* heuristic search planner, is proposed to exploit learned activity schemas as heuristics to make plans in similar situations in experience-based planning domains [17], and it is applied to solving problem instances of the same task with varying sets of objects [11]. However, using a plan generated by SBP to handle other situations such as an environment that changes dynamically during the task execution has not been reported until now. Furthermore, the solution to planning a task that contains loops and multiple activity schema instances is not optimal.

Robots performing complex tasks in a dynamically changing environment involve multiple-tasks planning and replanning. Zhang [18,19] expanded the model of HTN planning with action cliques, and proposed an optimization method to improve the task execution efficiency. In the same work, when the protected preconditions or protected states fail, the execution monitor triggers partially backtracking replanning. Helen introduced an enhanced robotic planning framework that improved robots' ability to operate in dynamically changing environments [20]. However, these methods rely on the definition and description of a complete task decomposition method, and there is no evidence that these methods support abstracted planning experience.

In this paper, we present a novel learning and planning method that integrates the empirical activity decomposition schemas of different situations and generates optimal HTN plans through a partially backtracking algorithm and merging optimization. The robot system first gathers the robot activity experience of the teaching task. Secondly, the system learns the activity schema and combines it with the historical activity schemas. The process consists of three steps: extracting the predicates, which is the component of the abstract method preconditions; determining the structure of the new activity schema which consists of abstract methods (see Definition 2); and merging the activity schema with the historical activity schema by changing its structure. Finally, to improve the task planning and execution efficiency in the process of generating a task plan based on activity schema, the actions merging the optimization solver and the partially backtracking solver are added

to the planner. The proposed learning and planning method is demonstrated on the Robustness by Autonomous Competence Enhancement (RACE) platform [21], and on our robot through two task scenarios: "deal with obstacles in serving a coffee to a guest" and "collect and deliver drink boxes to a counter". The results demonstrate that our method enables robots to better organize and utilize the various task experiences previously learned to solve the planning and optimization problems of similar tasks in a changing environment.

In this paper, we make the following contributions:

- We introduce abstract methods to integrate the empirical activity schemas of multiple situations, allowing the robot to adapt task plans in a dynamically changing environment by selecting the branches that correspond to learned activity schema from a similar situation.
- We present a novel algorithm that learns an activity schema with abstract methods for scenarios with complex tasks.
- We present an approach of generation of HTN plans based on integrated empirical activity schema with merging optimization, which solves the tasks with loops in the plan layer.
- We present a replanning algorithm that partially backtracks to the parent node layer by layer in the activity schema tree.
- We demonstrate the proposed algorithms with simulation and physical experiment in two scenarios.

The paper is structured as follows. We first present the related work, overall learning and planning architecture in Section 1. Section 2 proposes learning algorithms for multiple activity schemas with the abstract methods, and a planning algorithm with optimization. Finally, the respective experiments in the simulations of restaurant scenarios and in the physical robot of the delivery domain are demonstrated and compared with the state-of-the-art methods in Section 3, and the conclusion and discussion of future studies are summarized in Section 4.

2. Learning and Planning Methods

2.1. Overall Framework

In this section, we present the modified components of the modular RACE from the learning and planning perspective as red components in Figure 1. The central component is working memory. The basic data type of data written to the working memory is fluent [21]. The user interface is used to send a planning goal to the working memory which triggers the Task Planner and teaches a robot how to perform a complex task. The experience extractor continuously observes the history of fluents in the working memory to detect and extract potential relevant experiences including the sequence of applied actions, the methods with names and arguments, and user feedback. Taking the recorded plan-based robot activity experiences and learned activity schema as input, the activity schema learner constructs and records an activity schema with abstract methods into the activity schema memory. The developed task planner is composed of three components: a schema-based planner, merging optimization solver and partially backtracking solver. The merging optimization solver is employed to find an optimal solution to a given plan generated by the schema-based planner. The planner creates a plan as a sequence of ground actions and writes the plan into the working memory. The execution manager receives the plan, dispatches the planned actions to the robot, and records success or failure information into the working memory. If an exception happens, the planner will replan using the partially backtracking solver. The activity schema learner and task planner are addressed in this paper.

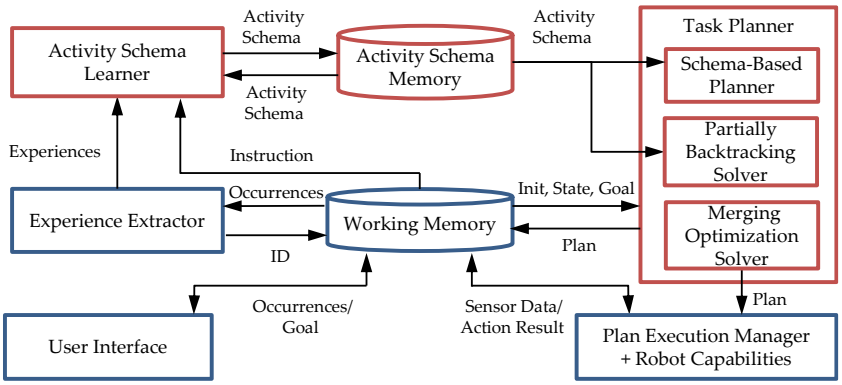


Figure 1. An overview of the learning and planning system.

2.2. Representation

In this section, we propose or modify some definitions in experience-based planning domains (EFPDs). The representations of actions can be hierarchical as in HTNs [22]. The main definitions of main concepts in this paper are listed below:

Definition 1. A **method**: $m = (h, L_1\ T_1, L_2\ T_2, \ldots\ L_n\ T_n)$, in which h is the abstract method's head including the name of an argument list for m, L_i is a predicate list for the preconditions of the ith branch, and T_i is a subtask list. Based on methods, the planner knows how nonprimitive tasks can be decomposed into other tasks.

There is a complete definition of methods and operators in the complete task knowledge domain, and the methods are used to guide the decomposition of a task under multiple conditions as "middle layers". Inspired by this notion, to merge the plan-based activity experiences from multiple situations in the same task, we define the abstract methods that served as middle layers in the merged activity schema.

Definition 2. An **abstract method**: $am = [h, (presub_1)(sub_1), (presub_2)(sub_2), \ldots, (presub_n)(sub_n)]$, where h is the head of the abstract method, $(presub_i)(sub_i)$ is the ith branch of am, $presub_i$ is a predicate list for the preconditions of the ith branch, and sub_i is a list of ordered subtasks, which is composed of abstract operators or other abstract methods.

Definition 3. An **activity schema** is a triple, $ma = (t, AM, \Omega)$, where t is a task head, AM is the set of abstract methods to decompose the t and Ω is an enriched abstract plan to achieve the t. Here, ma is a task model learned from an experience.

The research goal is how to merge activity experiences from multiple situations and generate an intelligent and efficient plan that could adapt to the dynamically changing environment. There are four research problems in this paper and the respective solutions are given as follows.

Problem 1: How to abstract the sequence of abstract methods AM from taught subtasks sequence M, and conceptualize the activity experiences as activity schema ma. Problem 1 is formalized as: $AM = \text{abs}(M)$, $ma = (t, AM, \Omega)$.

Solution 1: The preconditions and structures of the abstract methods AM that compose the activity schema ma can be extracted according to the function abstract (M) in Algorithm 1 in Section 2.3.

Problem 2: How to merge activity experiences for multiple situations through abstract methods. Let ma be an activity schema for the target task t, and ma' is a merged activity schema in the activity schema memory for the identical target task t. Problem 2 is formalized as: $ma' = \text{mer}(ma, ma')$.

Solution 2: The preconditions or structures of the abstract methods *MA'* that compose the activity schema *ma'* can be detected and adjusted according to the function Merge (*ma*, *ma'*) in Algorithm 1 in Section 2.3.2.

Problem 3: How to find an optimal plan for a learned task with different objects. Problem 3 is formalized as: (π, *tasktree*, *plantree*) = *P*(*D*, *P*, *ma*), where π is an optimal plan solution, *tasktree* is a task tree [19], *plantree* is a plan tree [19], *P* is a task planning problem, and *D* is an experience-based planning domain [2].

Solution 3: Planning based on activity schema with merging optimization as the function SBOP(*D*, *P*, *ma*) in Algorithm 2 in Section 2.4.1.

Problem 4: How to replan automatically to recover from the failure that is due to an exception that has been merged in the activity schema. Problem 4 is formalized as: (π, *tasktree'*, *plantree'*) = *REP*(*S*, *failedAction*, *ma*, *tasktree*, *plantree*), where *tasktree'* is the updated task tree, *plantree'* is the updated plan tree, *S* is the current world state, and *failedAction* is the failed action.

Solution 4: Planning based on activity schema with partially backtracking technique as the function replanning_pb(*S*, *failedAction*) in Algorithm 3 in Section 2.4.2.

2.3. Learn Activity Schema with Abstract Methods

Our goal is to generalize and conceptualize the collected robot activity experience with activity schema, so we need to extract the preconditions and structures of the abstract methods that comprise the activity schema.

2.3.1. Relevant Predicates Extraction

The first problem we try to solve is how to extract the predicate list of an abstract method from the robot's experience. The number of predicates in a robot system with complex scenes and tasks is large, but the predicates in preconditions of a method are limited. We found that most predicates in a precondition of a method can be grouped into two categories. The first category contains one of the subtask's arguments and the second category contains another argument of a predicate belonging to the first category.

In the property graph for a concept, nodes within a semantical distance of 2 are considered as relevant [23]. Given that the goal of a task is to change the state of some objects, the objects used as arguments of the taught task, the preconditions and effects of actions are considered as the **key** arguments. The predicates that connect an argument of a subtask with a key argument within a two-step relationships are considered as relevant predicates and extracted as the precondition of the taught task (an activity schema or an abstract method). In the system, all predicates follow one of the following two forms: (Instance class instance_name) and (property instance_name filler). The former states the existence of an instance of the class type and the latter is used to represent the properties of that instance.

Let L_1 be the predicates following the first form, and let L_2 be the predicates following the second form. The relationship from one argument x to another parameter y is calculated by the number of predicates connecting them that follow the second form, and the sign of the relationship is determined by the direction. However, if the number of connected predicates is 2, and the intermediate argument is reified relation (e.g., at, on), the distance is considered as 1 (as in Formula (3)). Additionally, a predicate in the form of (Instance class instance_name) that contains an argument x is regarded as distance 0. According to such rules, the relevant predicates for a branch b of an abstract method are computed as follows:

$$F(b) = \cap_{i=1}^{n} F(t_i), \tag{1}$$

$$F(t) = F_0(t) \cup F_1(t) \cup F_2(t) \cup F_{-1}(t) \cup F_{-2}(t), \tag{2}$$

$$F_1(t) = \{\tau(p(x, y)) \mid p \in L_2,\ x \in A,\ y \in B\} \cup \{\tau_1(p(x, z)),\ \tau_2(q(z, y)) \mid (p, q) \in L_2,\ x \in A,\ y \in B,\ z \in R\}, \tag{3}$$

$$F_{-1}(t) = \{\tau(p(y,\ x)) \mid p \in L_2,\ x \in A,\ y \in B\} \cup \{\tau_1(q(y,\ z)),\ \tau_2(p(z,\ x)) \mid (p,\ q) \in L_2,\ x \in A,\ y \in B,\ z \in R\}, \quad (4)$$

$$F_2(t) = \{\tau_1(p(x,\ z)),\ \tau_2(q(z,\ y)) \mid (p,\ q) \in L_2,\ x \in A,\ y \in B,\ z \notin R\}, \quad (5)$$

where t_i is the ith subtask in b; p and q are predicates; τ, τ_1 and τ_2 are qualitative timestamps; A is the set of arguments of a task t; B is the set of key arguments; and $F_i(t)$ is the collection of predicates that connect x with y with distance i. R is a set of arguments that are reified relations.

To support similarity-based planning, the relevant predicates are also used as features for each abstract operator via the same calculation as mentioned above. In the RACE system, propositions in the form of first-order binary predicates with qualitative time stamps are called fluents, and are gathered by an experience extractor.

2.3.2. Determine the Structure of the Abstract Method

In our system, the robot has basic action knowledge consisting of methods and operators. In general, there are two cases that the robot needs to learn from: 1) the first time when the robot faces a new task consisting of the subtasks sequence provided by a human teacher; 2) a failure situation of an existing task happens due to the learned activity schema not being applied, and a subtasks sequence is provided to recover from the failure situation.

For the purpose of formulating general concepts from a single training example and domain knowledge, deductive generalization is employed [8]. To improve the adaptability and flexibility of conceptual experience (activity schema), an abstract method is used as the middle layer. The learning algorithm is elaborated in Algorithm 1, which goes through the following steps:

Algorithm 1 Learning activity schema with abstract methods

1: generalize a robot activity experience e using EBG
2: abstract and translate operators to enriched abstract operators Ω
3: $AM \leftarrow$ abstract (M)
4: | **for** each m_i in M **do**
5: | name(am_i) $\leftarrow$ name(m_i); argument(am_i) $\leftarrow$ argument(m_i)
6: | subtasks of $am_i \leftarrow$ sequence of enriched abstract operators Ω_i under am_i
7: | preconditions of $am_i \leftarrow F(\Omega_i)$
8: | $F(AM) \leftarrow F(M)$
9: $ma \leftarrow (t, AM, \Omega)$
10: Merge(ma, ma')
11: | **if** $AM' \neq AM$ **then**
12: | **if** $AM' \subset AM$ **then**
13: | $AMF \leftarrow$ add generalize($AM' - AM$) ; **return**
14: | **else**
15: | add *subtask* to the ma under the preconditions $F(AM)$
16: | $F(AM') \leftarrow$ add not($F(AM)$ - $F(AM')$); **return**
17: | $F(AM') \leftarrow F(AM) \cap F(AM')$
18: | **for** each am_i in ma **do**
19: | **for** k = 1 to num(sub_k')
20: | **if** $sub_k' = \Omega_i$ **then** $pre\text{sub}_k' \leftarrow F(\Omega_i) \cap presub_k'$; **return**
21: | **elseif** $sub_k' \subset \Omega_i$ **then**
22: | $amf \leftarrow$ generalize ($\Omega_i - sub_k'$) under the preconditions $F(\Omega_i)$-$F(sub_k')$
23: | $AMF \leftarrow$ add amf ; **return**
24: | add Ω_i to the am_i under the preconditions $F(\Omega_i)$; num(sub_k') $\leftarrow$ num(sub_k')+1
25: | **for** k = 1 to num(sub_k')-1 **do**
26: | $F(\text{sub}_j') \leftarrow$ add not($F(\Omega_i)$ - $F(sub_j')$)
27: | **return** ma'

1. Generalize a robot activity experience e using explanation-based generalization (EBG) as the work [2].
2. The generalized operators are abstracted and translated to enriched abstract operators Ω [2] with features.

3. In the first case, the taught subtask sequence M can be directly transformed into a corresponding abstract method sequence AM (line 3). The name and arguments of the abstract method are the same as the corresponding subtasks (lines 5–6), and each abstract method consists of a sequence of corresponding abstract operators (line 7). In this way, the initial three-layer structure of an activity schema is built: the schema layer, the abstract method layer and the abstract operator layer.
4. In the second case, the corresponding abstract method sequence AM is obtained using the previous method. If AM' (original abstract method sequence) is a true subset of AM, the complement of set AM' in set AM will be extracted as a free abstract method and generalized as AMF (line 13). The generalization process involves replacing the observed constants in the arguments of abstract methods, operators and their preconditions with variables. If AM does not contain the same sequence as AM', it is created as a new branch of the activity schema ma under the preconditions $F(AM)$ (line 15), and preconditions of other branches will be adjusted as noted in line 16. Otherwise, the preconditions of AM and AM' (AM is the same as AM') will be merged, and the abstract operator sequence Ω_i under each am_i should be checked. Three potential cases are noted: 1) if an identical abstract operator sequence sub_k' as Ω_i exists in the branches of the abstract method in ma', the preconditions of them will be combined by intersection (line 20); 2) if a branch sub_k' which is a subset of Ω_i exists, the complement of set sub_k' in set Ω_i will be extracted as a free abstract method and generalized as amf; and 3) otherwise, Ω_i will be regarded as a novel branch and added to am_i under the preconditions $F(\Omega_i)$ (line 24), and preconditions of other branches will be adjusted as noted in line 26.

2.4. Planning Based on Activity Schema with Merging Optimization and Partially Backtracking Techniques

2.4.1. Planning Based on Activity Schema with Merging Optimization

To find an optimal solution to a task planning problem P in a given domain D, we developed a schema-based optimized planner (SBOP) which is based on the learned activity schema, and actions merging optimization and partially backtracking techniques are adopted.

The SBOP is presented in Algorithm 2. SBOP begins via selecting an activity schema ma with respect to a given task t. After instantiation, a global task tree rooted as '*tastree*' is created to store the task tree, and a global plan tree rooted as '*plantree*' is created to store the action chains that implement the task goal [19].

Given that the execution of the lowest-level actions changes the state of the world and affects the judgment of its subsequent nodes' preconditions, the check state of the current node depends on the check state of its children nodes. The search algorithm is designed as a depth-first algorithm. Only when all the children nodes are checked, is the next node in the same branch checked. When all the nodes in the current branch are checked, the parent node is returned to check the next node of the parent node.

We set three values for the check state of an abstract method instance node: "0" indicates that the nodes in its branch have not been checked, "1" indicates that all the nodes in one of its branches have been detected, "2" indicates that it is being checked. The abstract method am is checked layer by layer according to the structure of the selected activity schema. In this process, the current world states are substituted into the preconditions of each branch in am to check whether sub_i is applicable in a state S (sub_i is applicable in a state S only if there is a binding θ that binds all variables in the corresponding task's arguments [12], referring to line 19). If there is such a branch, the subtasks $sub_i\theta$ of *activenode* are added to *tasktree*. Then, the check state of *activenode* is set to "2", and the nodes in the $sub_i\theta$ are checked one by one (lines 21–22). If such a branch does not exist, the preconditions that are not satisfied will be published. If *activenode* is an instance of an abstract operator, then S is updated to S' by applying active action to the transition function. If S' is true, the current action a is added to *plantree* and *tasktree* (line 30).

The condition for terminating a loop in this check is that a branch of the current activity schema is traversed, that is, state(*ma*) = 1. It is very likely that the task contains multiple subtasks that conform to the same schema. It is necessary to determine whether the search should be terminated by judging the satisfactory conditions in the goal proposition in *S*′. Finally, the planner stops and returns the *tasktree* and *plantree*.

To improve the execution efficiency, the whole plan is subject to merging optimization, and the action cliques [19] are generated by selecting the actions layer by layer of the plan tree in a top-down manner until the final plan is obtained (line 34). The optimization is modelled as a MWCP (maximum-weight clique problem) [24] based on the graph theory. Finally, the optimized plan is taken as the output.

Algorithm 2 Schema-Based Optimized Planner (SBOP)

Procedure SBOP(*D*, *P*, *ma*)
1: | (*t*, *AM*, Ω) ← select an activity schema *ma*(*t*, *AM*, Ω)∈*MA* for task *t*
2: | **instantiate**(*t*, *P*)
3: | *activenode* = root node of *ma*; state(*ma*) ← 0
4: | **while** state of state(*ma*) ≠ 1 **do**
5: | | **if** *activenode* is an instance of abstract method *am*∈*AM* **then**
6: | | | **if** state(*activenode*) = 2 **then**
7: | | | **for** each node n_i in sub_i of *activenode* ***do***
8: | | | **if** state($\boldsymbol{n_i}$) = 1 and state($\boldsymbol{n_{i+1}}$) = 0 **then**
9: | | | *activenode* ← $\boldsymbol{n_{i+1}}$; **return**
10: | | | **if** state($\boldsymbol{n_i}$) = 1 and $\boldsymbol{n_{i+1}}$ =φ **then**
11: | | | state(*activenode*) = 1; **return**
12: | | | **return**
13: | | | **if** state(*activenode*) = 1 **then**
14: | | | **if** next node of *activenode* belonging to the same branch is φ **then**
15: | | | state(parent(*activenode*)) = 1; **return**
16: | | | **else** *activenode* ← next node of *activenode*; **return**
17: | | | **if** state(*activenode*) = 0 **then**
18: | | | get sharing states and update *S* in *P*
19: | | | check ← whether there exist sub_i in *activenode* applicable in a state *S*
20: | | | **if** check is false **then** publish the preconditions which are not ready
21: | | | **else break**
22: | | | *tasktree*.add_children($sub_i\theta$)
23: | | | state(activenode) ← 2; activenode ← head($sub_i\theta$)
24: | | | **return**
25: | | **else**
26: | | | **if** state(*activenode*) = 0 **then**
27: | | | get sharing states and update *S* in *P*
28: | | | pick the action *a* applicable to *S* with the lowest cost using SBP
29: | | | *S*′← γ(*S*, *a*) ; transition function
30: | | | **if** *S*′; is true **then**
31: | | | *tasktree*.add_child (*a*); *plantree.add_child* (*a*); state(*activenode*) ← 1;
return
32: | | | **return** failure
33: | **if** state(*ma*) =1 and goal proposition is not satisfied in *S*′**then** return to line 3
34: | **if** state(*ma*) =1 and goal proposition is satisfied in *S*′**then**
35: | translate *plantree* to *plantree* (*AC*) based on action relations analysis
36: | a plan π ← Optimize the *plantree* (*AC*)
37: | **else return** failure

2.4.2. Planning Based on Activity Schema with Partially Backtracking Technique

When execution fails, the replanning is activated to recover from the failure. If the failure is due to an exception that has been merged in the activity schema, the recovery will be successful. The process is presented in Algorithm 3.

The failed action in a *tasktree* can be obtained by Action2Task(), which maps the actions from the *plantree* to *tasktree* according to a dictionary (line 2). Replanning is conducted for the task tree under the parent node *failedParent* of the failed action. We first translate *failedParent* to the corresponding abstract method *am* in the activity schema, and check whether the branch $presub_i$ in *am* exists that is applicable in state *S* (lines 5–6). If no such branch is identified, we will check whether there exists amf_i in *AMF* applicable in state *S*. If not, the parent node of *failedParent* will continue to be translated and checked (line 11). If an amf_i exists, the children nodes of the branch $amf_i\theta$ will be added to the node *failedParent* in the task tree (line 13). If $presub_i$ exists, the first abstract operator ω_f whose feature with qualitative timestamps "at_goal" (refer to [2]) is satisfied under sub_i should be identified, and the children nodes of *failedParent* in the task tree will be replaced by the nodes of the branch $sub_i\theta$ after $\omega_f\theta$ (line 16). After that, the plan solution can be found according to SBOP. Figure 2 presents the basic principle.

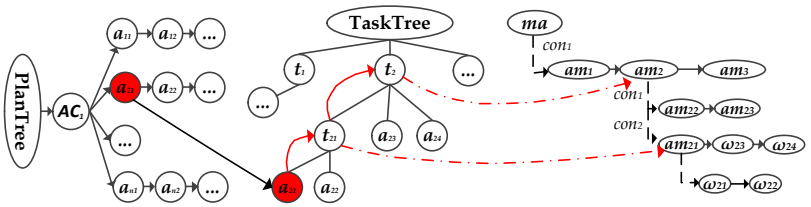


Figure 2. Basic principle of replanning based on partial backtracking. Red nodes are the failed actions. Red solid arrows represent the directions of a search; red dotted arrows are the function abstracts.

Algorithm 3 Replanning Based on Partially Backtracking

replanning_pb (*S*, *failedAction*)
1: | solution ← *False*
2: | failedTask ← Action2Task (failedAction)
3: | failedParent ← failedTask.up
4: | **while** solution = *False* **do**
5: | | *am* ← abstract (*failedParent)*
6: | | check ← whether there exist $presub_i$ in *am* applicable in a state *S*
7: | | **if** check is false **then**
8: | | check ← whether there exist amf_i in *AMF* applicable in a state *S*
9: | | **if** check is false **then**
10: | | **if** *failedParent* is the root of *tasktree* **then return failure**
11: | | failedParent ← failedParent.up; **return**
12: | | **else**
13: | | *tasktree*.add_children($amf_i\theta$); **go to** line17
14: | | **else**
15: | | find the abstract operator ω_f whose feature is satisfied under sub_i
16: | | *tasktree*.delete_children(*failedParent*); *tasktree*.add_children($sub_i\theta$ after $\omega_f\theta$)
17: | | state(activenode) ← 2; activenode ← head(sub $_i$)
18: | | search the plan solution according to SBOP until to state(*ma*) =1
19: | | solution = *True* ; **return**
20: | translate *plantree* to *plantree* (*AC*) and optimize the *plantree* (*AC*)

3. Demonstration and Evaluation

The proposed approach is demonstrated in two different scenarios: "deal with obstacles in serving a coffee to a guest" for learning and planning based on activity schema with partially backtracking technique, and "collect and deliver drink boxes to a counter" for merging optimization of a plan.

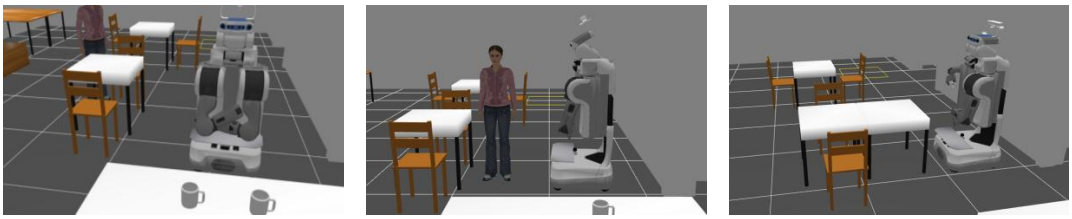
3.1. Demonstration 1: Simulation in Scenario "Deal with Obstacles in Serving a Coffee to a Guest"

3.1.1. Learning Abstract Methods and Robot Activity Schema

To allow the user to provide instructions to the robot, we use a command-line interface that supports the following three types of commands: "achieve" is used to specify a task to be performed,

"abort" is used to abort the requested task by the user, and "teach_task" is used to teach the robot a new task. Before performing or learning a new task, the operators and methods such as drive_robot, put_object, grasp_object and so on have been defined in JSHOP2 syntax.

We employed three sub-scenarios on the PR2 platform in a simulated restaurant environment to teach the "serve-a-coffee" task [25,26]. In the first sub-scenario, we teach the robot to serve a coffee to a guest from the right without any obstacle as shown in Figure 3a. In the second sub-scenario, an obstacle is present, which is a person in the manipulation area blocking the path to the south of Table 1; the robot was taught to wait for a while until the person leaves the region (as shown in Figure 3b). In the third sub-scenario, another type of obstacle, namely an extension table, blocks the manipulation area. The robot was taught to detour to the other side of Table 1 and place the coffee to the left of the guest (as shown in Figure 3c).



(**a**) (**b**) (**c**)

Figure 3. (**a**) Sub-scenario 1: learn how to serve coffee without any obstacle; (**b**) sub-scenario 2: learn how to deal with the obstacle, which is a person in this example; (**c**) sub-scenario 3: learn how to deal with the obstacle, which is an extension table in this example.

The learned activity schemas without the abstract method are presented in Figure 4, in which ma_1, ma_2 and ma_3 represent the learned activity schema in sub-scenarios 1 through 3, respectively. In ma_2, the abstract operator sequence in green is the part newly learned from sub-scenario 2, which is used to handle the problem of being blocked by a person. In ma_3, the abstract operator sequence in orange is the part newly learned from sub-scenario 3 to handle the case being blocked by a table. As a contrast, the structure and learned activity schema with abstract methods (denoted as *ma*) is presented in Figure 5. The key components of *ma* are shown in Figure 6.

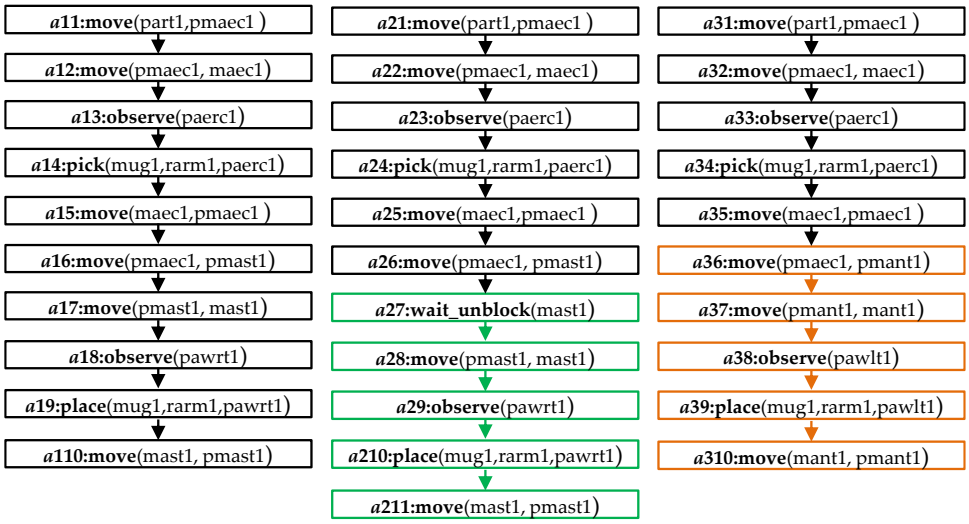


Figure 4. The activity schemas learned from the three sub-scenarios. The abstract operator sequences of ma_1, ma_2 and ma_3 are presented from left to right. Each abstract operator is represented by a rectangular box, in which "*ax*" is its abbreviation. The green part indicates a sequence in ma_2 that differs from ma_1. The orange part indicates a sequence in ma_3 that differs from ma_1.

Of note, ma_1, ma_2 and ma_3 have been integrated into *ma* owing to the middle layer of the abstract method. The initial three-layer structure of *ma* is learned from the first sub-scenario, and its abstract method sequence is recorded as *AM'* (line 8 in Listing 1). The sequence of abstract methods *AM2* extracted from sub-scenario 2 is noted as follows: [drive(part1, pmaec1), grasp(mug1, rarm1), drive(pmaec1, pmast1), wait_unblock(mast1), put(mug1, pawrt1)]. Clearly, $AM' \subset AM2$, so the complement of set *AM'* in set *AM2*, wait_unblock(mast1), is extracted as a free abstract method and generalized as wait_unblock- (?manArea) (lines 15–17 in Listing 1). Given that the sequence of abstract methods extracted from sub-scenario 3 did not contain the same sequence as *AM'*, it was created as a new branch of *ma* under the preconditions *F*(*AM*) (lines 10–12 in Listing 1), and preconditions of the other branch were adjusted (line 6 in Listing 1).

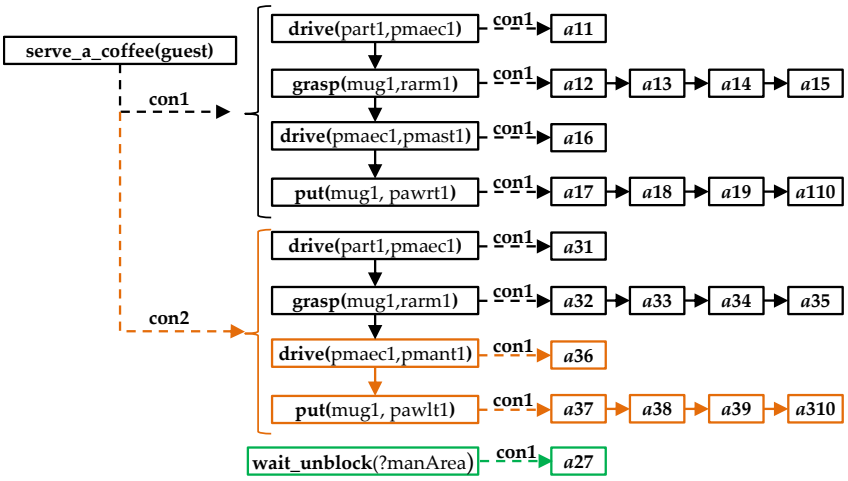


Figure 5. Structure of an activity schema with abstract methods *ma*. Dotted arrows indicate preconditions, and solid arrows indicate the order of execution. The middle rectangles represent the abstract methods. For the meaning of other rectangles and colours, refer to Figure 4.

1. [**serve_a_coffee** (guest1))
2. ((condition1;
3. (Instance PlacingArea paerc1) (Instance Counter counter1)(HasPlacingArea counter1 paerc1)
4. (Instance At at1)(HasPhysicalEntity at1 guest1)(HasArea at1 saw1)
5. (Instance SittingArea saw1)(**HasPlacingAreaRight** saw1 **pawrt1**) (Instance PlacingArea **pawlt1**)
6. **not**((hasArea **blockingAt1** mast1)(hasPhysicalEntity blockingAt1 obstacle1)(Instance Table obstacle1))
7.)
8. [**drive**(part1,pmaec1), **grasp**(mug1,rarm1), **drive**(pmaec1,pmast1), put(mug1, **pawrt1**)])
9. ((condition2;
10. . . . . (Instance SittingArea saw1)(**HasPlacingAreaRight** saw1 **pawrt1**)
11. (hasArea **blockingAt1** mast1)(hasPhysicalEntity blockingAt1 obstacle1) (Instance Table obstacle1)
12. (HasPlacingAreaLeft saw1 **pawlt1**) (Instance PlacingArea **pawlt1**))
13. [**drive**(part1,pmaec1) **grasp**(mug1,rarm1) **drive**(pmaec1,pmast1) put(mug1, **pawlt1**)])
14. Free abstract method;
15. ((Instance At ?at)(hasArea ?at ?manArea)(hasPhysicalEntity ?at ?obstacle)
16. (Instance Person ?obstacle))
17. [**wait _unblocked** (?manArea)]]

Listing 1. Key components of *ma*.

3.1.2. Planning Based on Activity Schema with Abstract Methods

To assess the performance of the planning algorithm based on activity schema with abstract methods, two types of obstacles, a table and a person, were both added to sub-scenario 4. The position of the table was the same as the scenario of learning, but the position of the person changed to the other

side of Table 1. The robot was not aware of the presence of obstacles at the beginning. The execution process of the plan generated by SBOP and SBP is presented in Figure 6.

Initially, a plan was generated following Algorithm 2. When the action with the highest percentage of verified features in *a*17 (move(pmast1, mast1)) was to be performed, the robot found that mant1 was occupied by a table, and the robot cannot complete the action. The failed execution of the action was published to the working memory, and when the planner received the fault information, replanning was activated to recover from the failure. *failedParent*, the parent nodes of the failed action in the task tree, was translated to the corresponding abstract method put(mug1, pawrt1) in the activity schema, and no such branch whose preconditions were applicable in state *S* was identified. Then, the parent node of *failedParent* continued to be translated to the corresponding abstract method, [drive(part1,pmaec1), grasp(mug1,rarm1), drive(pmaec1,pmast1), put(mug1, pawrt1)] (as shown in List 1), and the robot checked that the preconditions or condition 1 in List 1 were not applicable in state *S*. Then, the parent node, serve_a_coffee (guest1), continued to be checked, and preconditions of the second branch or condition 2 were applicable in state *S*. The first abstract operator *a*36 (move(pmaec1, pmant1)), whose feature with qualitative timestamps "at_goal" was satisfied under the second branch, was identified. So far, the best recovery position for the plan has been identified. Then, the children nodes of serve_a_coffee (guest1) in the task tree were updated from *a*36 as the Algorithm 2 and Algorithm 3, and a new optimal plan was generated after the merging optimization as the Algorithm 2.

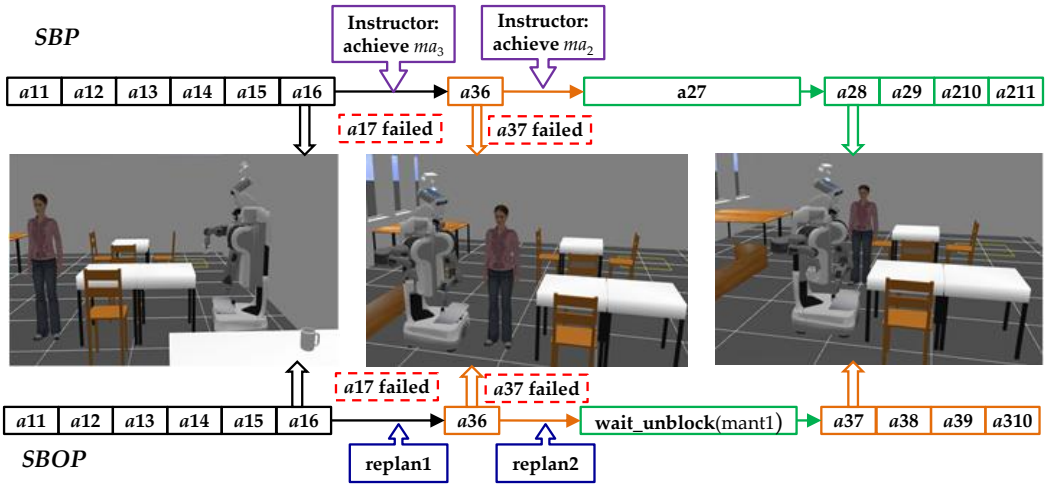


Figure 6. The execution process of the abstract plan based on schema-based planner (SBP) and schema-based optimized planner (SBOP). The purple boxes above the plan are abbreviated intervention instructions from the instructor. Each blue box under the plan is a process of replanning based on partially backtracking.

When the robot encountered the person who blocked the path towards Table 1, the free abstract method wait_unblock (?manArea)) was identified and translated to wait_unblock (mant1) through a binding "mant1". Then, the plan solution was successfully generated following Algorithm 3. The simulation result shows that, when using our replanning method and the learned activity schema with abstract methods, the robot's execution recovers from two failures automatically.

By contrast, when using traditional SBP and learned activity schema without an abstract method, and if *a*17 and *a*37 failed, the plan would fail because SBP does not have the ability to update the task problem and replan automatically. The instructor needs to designate the activity schema to guide the planning. This process is detailed below.

When the robot was to perform the action with the highest percentage of verified features in *a*17 (move(pmast1, mast1)) according to the plan π_1 generated based on ma_1, the robot cannot complete the action due to an unexpected obstacle table occupying mast1. However, the SBP does not have the ability to update the task problem and replan automatically. Therefore, when the current action fails,

the subsequent actions in π_1 could not be executed, and π_1 was abandoned due to the unreachability of the target state. As a result, the request was made to the instructor. At this time, the instructor was required to guide the robot. The instructor assigned the current activity schema to ma_3 according to the updated world state. Then, the SBP searched for a new plan—π_2 based on ma_3. The first action to be performed was move_base(pmast1, pmant1), which had the highest percentage of verified features in $a36$ (move(pmaec1, pmant1)). That meant that the robot should place the coffee cup on the left side of the guest (the north side of Table 1).

When the action with the highest percentage of verified features in a37 (move(pmant1, mant1)) was to be performed, the robot found that mant1 was occupied by a person, and the robot could not complete the action. Similarly, since the SBP could not update the task problem and replan automatically, the current action failed. When the follow-up action could not be performed, π_2 failed because the target state of the plan could not be reached, and the request was made again to the instructor. At this time, the instructor was required to guide the robot. The instructor assigned the current activity schema to ma_2 according to the updated world state. Then, the SBP searched for a new plan—π_3 based on ma_2. The first action to be performed was wait_unblock (mant1), which had the highest percentage of verified features in $a27$ (wait_unblock(mast1)). That meant that the robot should wait until the person leaves. Finally, the task was completed.

The performances of SBP and our planner SBOP were evaluated based on the aspects of autonomy, adaptability and efficiency. In terms of autonomy, SBP requires more than two intervention instructions during task execution, whereas SBOP does not require intervention instruction and can recover from failures automatically. In terms of adaptability, SBOP can adjust among multiple learned activity schemas during execution and generate the corresponding plans by detecting the preconditions. However, SBP does not have this capacity. In terms of efficiency, even though the plan-generating time cost for the first recovery of SBOP is greater than SBP's time to check the preconditions of the abstract methods, the total time for plan recovery from failure of SBOP is 59 seconds shorter. As shown in Table 1 below.

Table 1. Planners' performance in serve_a_coffee scenario.

Method	SBOP	SBP
Number of intervention instructions during execution	0	$\geq$2
Recover from the failure automatically	$\surd$	$\times$
Adapt among multiple learned activity schemas during execution	$\surd$	$\times$
Plan-generating time cost for the first recovery	205 s	149 s
Plan-generating time cost for the second recovery	223 s	205 s
Total time for plan recovery from failure	428 s	487 s

3.2. Demonstration 2: Physical Experiment in the Scenario "Collect and Deliver Drink Boxes to a Counter"

In this scenario, the environment is set with 3 tables (Table 1–3) and a counter (Table 4) distributed in a laboratory (as shown in Figure 7). The task for the robot is to move the drink boxes from 3 tables to the counter. Similar to PR2, the Inbot robot has learned an activity schema for collecting a beverage box, including several abstract methods, such as grasp_object, put and move_object. Inbot is a mobile robot equipped with an arm and a gripper, together with sufficient sensors for automated navigation and manipulation.

In the experiment, the action sequences generated with SBOP and SBP are presented with three colours in Figure 8. The actions in blue represent the same components in the two plans; the actions in orange represent components in the SBP-generated plan but removed in the SBOP-generated plan; and the actions in green represent the merged components in the SBP-generated plan compared with the SBOP-generated plan.

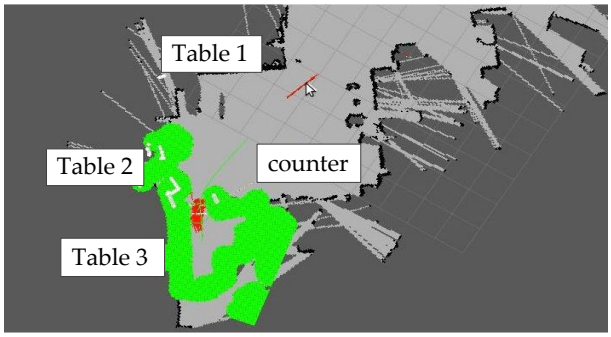
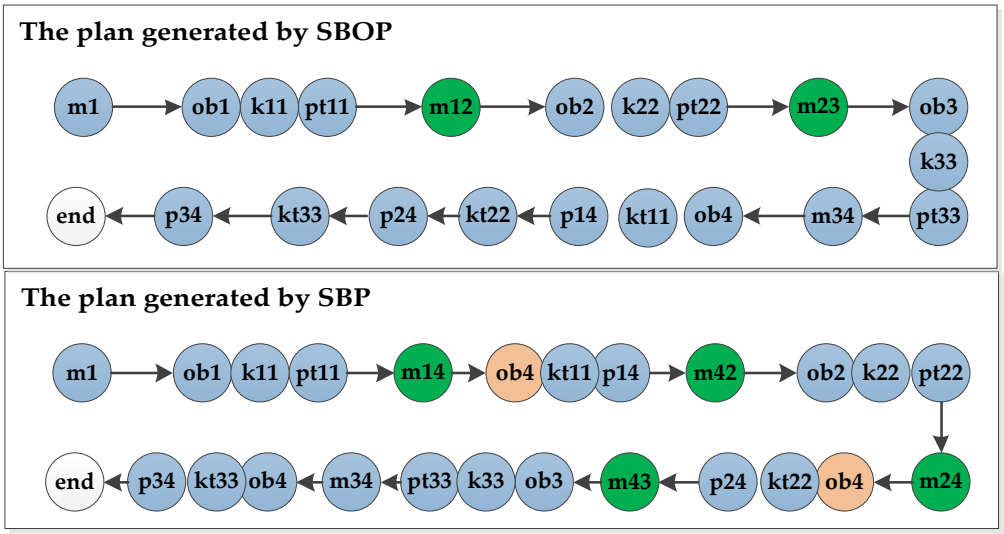


Figure 7. The experimental setup in a grid map.



Action example:
obx:!**observe_objects_on_area** (?plareaTablex) **mxy**: !**move_base**(Table x, Table y)
kxy: !**pick_up_object**(Drink x, Table y) **ktxy**: !**pick_from_tray**(Drink x, Trayarea y)
pxy: !**place_object**(Drink x, Table y) **ptxy**: !**place_on_tray**(Drink x, Trayarea y)

Figure 8. The action sequences generated by SBOP and SBP. The actions in blue represent the same components among the two plans; the actions in orange represent components in the SBP-generated plan but removed in the SBOP-generated plan; the actions in green represent the merged components in the SBP-generated plan compared with the SBOP-generated plan.

SBOP initially generated the same plan as SBP. Then, actions with independence or allowance relations were identified and placed in the same action clique. For example, since actions ob4 and kt11 exhibited independence relation, and kt11 and p14 exhibited allowance relation, the three actions could be placed in the same action clique AC_1 to complete the task of transferring drink box 1 to the counter in the order: (ob4, kt11, p14). If multiple drink boxes were on the robot tray, several action cliques, which were independent of each other, would be placed in a larger action clique AC. For example, when AC_1 (ob4, kt11, p14) and AC_2 (ob4, kt22, pt24) were placed in action clique AC, the repetitive action ob4 would be merged to leave only one in AC.

Then, in the optimization modeled as MWCP, the task plan with maximum time savings had the least number of actions, in which the robot collected the maximum number of the boxes at a time. During this process, two ob4 actions were deleted. Actions m14 and m42 were replaced by action m12, since m12 had the same preconditions as m14 and the same effect as m42, with shorter time costs. Actions m24 and m43 were replaced by action m23 accordingly.

The execution process with planner SBOP is shown in Figure 9. The performance comparison between SBOP and SBP is presented in Table 2. Although the plan-generating time of SBOP is 18.4%

longer than SBP, the execution time is 17.4% shorter. In total, the robot can save 178 s to complete all tasks with SBOP, indicating the improvement in efficiency of SBOP compared with SBP.

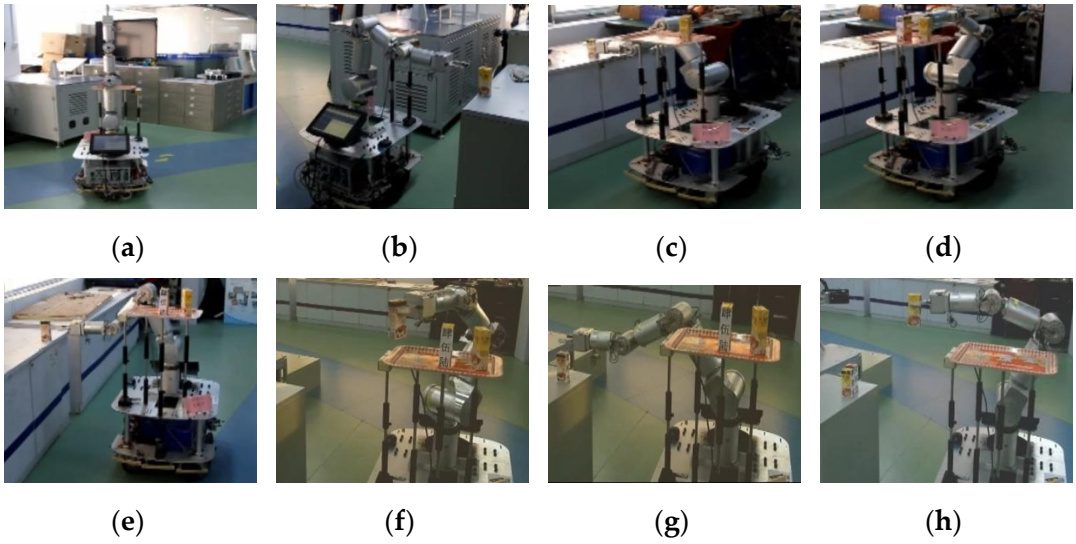


Figure 9. The execution process with the SBOP planner in the delivery domain.

Table 2. Planners' performance in scenario "collect and deliver drink boxes to a counter".

Planner	SBOP	SBP	Reducing Rate
Number of actions	21	25	16.0%
Cost time of execution	845 s	1023 s	17.4%
Cost time of planning	423 s	356 s	−18.1%
Total cost time	1268 s	1379 s	8.0%

4. Discussion and Conclusions

In this paper, the abstract method is proposed to serve as a middle layer for an activity schema to integrate the empirical activity schemas of multiple situations, and this method allows a robot to adjust task plans and adapt to the dynamically changing environment by selecting the corresponding branches. The method for extracting relevant predicates for preconditions of an abstract method proves to be simple and practical for scenarios with complex tasks and an uncertain environment. In addition, without relying on a large number of training examples, it can be used as a general method for determining preconditions of methods in case of insufficient training examples. The arguments and the names of abstract methods can be obtained from the methods previously defined directly. Thus, this method reduces the difficulty of learning and enhances the efficiency of applying the knowledge of the existing methods in not only the observed planning results but also the entire planning domain.

The developed planner SBOP supports task plan merging optimization and task replanning with a partially backtracking technique, which clearly provides better results compared with the state-of-the-art methods in terms of autonomy and adaptability, as shown in the simulation and physical experiment results. The integration of an activity schema-based planning and backtracking technique enables the robot to adjust the current plan under a changing environment, and search for coping strategies of similar situations from the experienced activity schema, thus demonstrating increased adaptability. Compared with the traditional partial backtracking technique, SBOP depends on action relations and the hierarchical structure of the task decomposition, which are provided by the abstract methods, rather than complete definition and description of a task. The integration of activity schema-based planning and merging optimization allows the generated plan to emerge from the possible suboptimal state and provide an optimal solution. To the best of our knowledge, the proposed learning and planning method is a novel solution of how to merge activity experiences

from multiple situations and generate an intelligent and efficient plan that could be adapted to a dynamically changing environment.

The computation load and time are significant for system learning and optimization. We considered three aspects of efficiency. Firstly, the efficiency of an activity schema-based planner will be improved with the increase of the number of task arguments [2]. Therefore, the instructor needs to provide sufficient task arguments to guarantee the efficiency of the plan. Due to the added abstract methods with additional task arguments in SBOP, the total number of task arguments of activity schema with abstract methods is often more sufficient than those without abstract methods, thus with higher efficiency. Secondly, using the merged activity schema in SBOP, the robot can automatically generate the corresponding plan when encountering similar situations. Without using abstract methods in SBP, the instructor needs to specify the activity schema in the similar situation. In contrast, automatic planning is more efficient. Thirdly, we introduced a partial backtracking technique in the replanning phase of SBOP, which not only realized automatic replanning but also improved efficiency compared to the method that required overall replanning.

However, there are several limitations to this research. The generalization performance remains to be improved. The efficiency of replanning is still relatively low even with partial backtracking, and certain issues still remain in terms of real-time replanning. One possible idea is to retain a few invalid supplement plans that can be switched to valid plans with minimal environmental change by designing an alternative layer in the planner [27].

Author Contributions: J.Z. proposed the idea of the paper; Y.H. and Z.C. conceived and designed the experiments. Z.C. performed the experiments. Z.C., B.Z. and S.Z. analyzed the data and wrote the paper. All authors have read and approved the final manuscript.

Funding: This work is financially supported by Shenzhen Key Laboratory Project (Grant No. ZDSYS20170727 1637577), in part by Shenzhen Peacock Plan (Grant No. KQTD20161130105710 19) and Shenzhen Fundamental Research Funds (Grant Nos. JCYJ20160608153218487, JCYJ20160229202315086 and JCYJ20170413104438332).

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Ingrand, F.; Ghallab, M. Deliberation for autonomous robots: A survey. *Artif. Intell.* **2017**, *247*, 10–44. [CrossRef]
2. Mokhtari, V.; Lopes, L.S.; Pinho, A.J. Experience-based planning domains: An integrated learning and deliberation approach for intelligent robots. *J. Intell. Rob. Syst.* **2016**, *83*, 463–483. [CrossRef]
3. Tenorth, M.; Klank, U.; Pangercic, D.; Beetz, M. Web-enabled robots. *IEEE Rob. Autom Mag.* **2011**, *18*, 58–68. [CrossRef]
4. Zablith, F.; Antoniou, G.; d'Aquin, M.; Flouris, G.; Kondylakis, H.; Motta, E.; Plexousakis, D.; Sabou, M. Ontology evolution: A process-centric survey. *Knowl. Eng. Rev.* **2015**, *30*, 45–75. [CrossRef]
5. Rockel, S.; Neumann, B.; Zhang, J.; Dubba, K.S.R.; Cohn, A.G.; Konecny, S.; Mansouri, M.; Pecora, F.; Saffiotti, A.; Günther, M. An ontology-based multi-level robot architecture for learning from experiences. In Proceedings of the AAAI 2013 Spring Symposium, Palo Alto, CA, USA, 25–27 March 2013; AAAI: San Francisco, CA, USA; pp. 2–4.
6. Martínez-Costa, C.; Schulz, S. Ontology-based reinterpretation of the snomed ct context model. In Proceedings of the International Conference on Biomedical Ontologies (ICBO 2013), Montreal, QC, Canada, 8–9 July 2013; pp. 90–95.
7. Lopes, L.S. Failure recovery planning in assembly based on acquired experience: Learning by analogy. In Proceedings of the 1999 IEEE International Symposium on Assembly and Task Planning (ISATP'99), Porto, Portugal, 21–24 July 1999; pp. 294–300.
8. Lopes, L.S. Failure recovery planning for robotized assembly based on learned semantic structures. In Proceedings of the IFAC International Workshops Intelligent Assembly and Disassembly (IAD'07), Alicante, Spain, 23–25 May 2007; pp. 65–70.

9. Lim, G.H.; Oliveira, M.; Mokhtari, V.; Kasaei, S.H.; Chauhan, A.; Lopes, L.S.; Tomé, A.M. Interactive teaching and experience extraction for learning about objects and robot activities. In Proceedings of the The 23rd IEEE International Symposium on Robot and Human Interactive Communication (IEEE RO-MAN 2014), Edinburgh, UK, 25–29 August 2014; pp. 153–160.
10. Mokhtari, V.; Lim, G.H.; Lopes, L.S.; Pinho, A.J. Gathering and conceptualizing plan-based robot activity experiences. In Proceedings of the 13th International Conference on Intelligent Autonomous Systems (IAS-13), Padova, Italy, 15–18 July 2014; pp. 993–1005. [CrossRef]
11. Mokhtari, V.; Lopes, L.S.; Pinho, A.J. An approach to robot task learning and planning with loops. In Proceedings of the 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2017), Vancouver, BC, Canada, 24–28 September 2017; pp. 6033–6038.
12. Ilghami, O.; Munoz-Avila, H.; Nau, D.S.; Aha, D.W. Learning approximate preconditions for methods in hierarchical plans. In Proceedings of the 22nd International Conference on Machine Learning (ICML 2005), Bonn, Germany, 7–11 August 2005; Luc, D.R., Stefan, W., Eds.; ACM: New York, NY, USA, 2005; pp. 337–344.
13. Xu, K.; Muñoz-Avila, H. A domain-independent system for case-based task decomposition without domain theories. In Proceedings of the 20th national conference on Artificial intelligence (AAAI 2005), Pittsburgh, PA, USA, 9–13 July 2005; AAAI: San Francisco, CA, USA, 2005; pp. 234–240.
14. Nejati, N.; Langley, P.; Konik, T. Learning hierarchical task networks by observation. In Proceedings of the 23rd International Conference on Machine Learning (ICML 2006), Pittsburgh, PA, USA, 25–29 June 2006; Cohen, W.W., Moore, A., Eds.; ACM: New York, NY, USA, 2006; pp. 665–672.
15. Hogg, C.; Munoz-Avila, H.; Kuter, U. Htn-maker: Learning htns with minimal additional knowledge engineering required. In Proceedings of the 23rd AAAI Conference on Artificial Intelligence (AAAI 2008), Chicago, IL, USA, 13–17 July 2008; AAAI: San Francisco, CA, USA, 2008; pp. 950–956.
16. Zhuo, H.H.; Hu, D.H.; Hogg, C.; Yang, Q.; Munoz-Avila, H. Learning htn method preconditions and action models from partial observations. In Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI 2009), Pasadena, CA, USA, 11–17 July 2009; Craig, B., Ed.; AAAI: San Francisco, CA, USA, 2009; pp. 1804–1810.
17. Mokhtari, V.; Lopes, L.S.; Pinho, A.J.; Lim, G.H. Planning with activity schemata: Closing the loop in experience-based planning. In Proceedings of the 2015 IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC 2015), Espinho, Portugal, 14–15 May 2014; IEEE: New York, NY, USA, 2014; pp. 9–14.
18. Chen, Z.; Zhang, J.; Hu, Y.; Dong, X.; Jin, H.; Zhang, J. Multitasking planning for service robot based on hierarchical decomposing. *ICIC Express LeTt. Part B, Appl.* **2016**, *7*, 77–83. [CrossRef]
19. Zhang, J.; Chen, Z.; Hu, Y.; Zhang, J.; Luo, Z.; Dong, X. Multitasking planning and executing of intelligent vehicles for restaurant service by networking. *Int. J. Distrib. Sens. Netw.* **2015**, *11*, 273825. [CrossRef]
20. Harman, H.; Chintamani, K.; Simoens, P. Architecture for incorporating internet-of-things sensors and actuators into robot task planning in dynamic environments. In Proceedings of the 2017 IEEE International Symposium on Robotics and Intelligent Sensors (IRIS2017), Ottawa, ON, Canada, 5–7 October 2017; IEEE: New York, NY, USA, 2017; pp. 13–18.
21. Meditskos, G.; Bassiliades, N. Rule-based owl ontology reasoning using dynamic abox entailment. In Proceedings of the 18th European Conference on Artificial Intelligence (ECAI 2008), Patras, Greece, 21–25 July 2008; Ghallab, M., Spyropoulos, C.D., Fakotakis, N., Avouris, N., Eds.; IOS Press: Amsterdam, Netherlands; pp. 731–732.
22. Erol, K.; Hendler, J.A.; Nau, D.S. Umcp: A sound and complete procedure for hierarchical task-network planning. In Proceedings of the 2nd International Conference on Artificial Intelligence Planning Systems (AIPS-94), Chicago, IL, USA, 13–15 June 1994; Kristian, H., Ed.; AAAI: San Francisco, CA, USA, 1994; pp. 249–254.
23. Neumann, B.; Hotz, L.; Rost, P.; Lehmann, J. A robot waiter learning from experiences. In Proceedings of the 10th International Conference on Machine Learning and Data Mining (MLDM 2014), St. Petersburg, Russia, 21–24 July 2014; pp. 285–299.
24. Östergård, P.R. A new algorithm for the maximum-weight clique problem. *Nord. J. Comput.* **2001**, *8*, 424–436. [CrossRef]

25. Zhang, L.; Rockel, S.; Pecora, F.; Hotz, L.; Lu, Z.; Klimentjew, D.; Zhang, J. Evaluation metrics for an experience-based mobile artificial cognitive system. In Proceedings of the 2013 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2013), Workshop on Metrics of Embodied Learning Processes in Robots and Animals, Tokyo, Japan, 3–7 November 2013; IEEE: New York, NY, USA, 2014; pp. 2225–2232.
26. Rockel, S.; Klimentjew, D.; Zhang, L.; Zhang, J. An hyperreality imagination based reasoning and evaluation system (hires). In Proceedings of the 2014 IEEE International Conference on Robotics and Automation (ICRA 2014), Hong Kong, China, 31 May–7 June 2014; IEEE: New York, NY, USA, 2014; pp. 5705–5711.
27. Hayashi, H.; Tokura, S.; Ozaki, F. Towards real-world htn planning agents. In *Knowledge Processing and Decision Making in Agent-Based Systems*; Jain, L.C., Nguyen, N.T., Eds.; Springer: Berlin, Germany, 2009; pp. 13–41.