A Forward Chaining Heuristic Search with Spatio-Temporal Control Knowledge

Xu Lu[1], Jin Cui[2(✉)], Yansong Dong[1], Wensheng Wang[1], Runzhe Ma[1], Yifeng Li[3], and Qing Feng[1]

[1] ICTT and ISN Lab, Xidian University, Xi'an 710071, People's Republic of China
[2] Xi'an Shiyou University, Xi'an 710065, People's Republic of China
cuijin_xd@126.com
[3] Xi'an University of Posts and Communications, Xi'an 710121, People's Republic of China

Abstract. Planning systems use a variety of heuristic search or control knowledge in order to enhance the performance. Control knowledge is often described in a specific formalism, e.g. temporal logic, automata, or HTN (Hierarchical Task Network) etc. Heuristic search exploits heuristic functions to evaluate potential feasible moves. Control knowledge constraints the search space by pruning the states which violate the knowledge. In this paper, we propose a general heuristic algorithm that combines control knowledge specified by a spatio-temporal logic named PPTL^{SL}. Both heuristic search and control knowledge are handled in a forward chaining manner. Our approach involves the evaluation of PPTL^{SL} formulas using a variant of the traditional progression technique during heuristic search. Consequently, we are enabled to take advantage of the two methods together to further reduce the search space.

Keywords: Planning · Control knowledge · Heuristic search · Forward chaining

1 Introduction

The planning problem in Artificial Intelligence is about the decision making performed by intelligent creatures like robots, humans, or computer programs when trying to achieve some goal. In particular, the most basic form of classical planning tries to find a sequence of actions leading from the initial state to a state which includes all goal constraints. Since the search space usually grows exponentially when the number of actions increases [8], search algorithms should be enhanced in order to guide the search appropriately. One of the best known method is using heuristic functions which estimate the distance from a given

This research is supported by the National Natural Science Foundation of China Grant Nos. 61806158, China Postdoctoral Science Foundation Nos. 2019T120881 and 2018M643585.

H. Miao et al. (Eds.): SOFL+MSVL 2019 Workshop, LNCS 12028, pp. 141–154, 2020.
https://doi.org/10.1007/978-3-030-41418-4_11

state to the nearest (or optimum) goal state. Another one is adopting control knowledge which describes additional information of a specific planning problem to constrain the search space.

On one hand, although a number of heuristic search algorithms are proposed for the last two decades, these algorithms are limited in that they are only suitable for certain kinds of planning problems. On the other hand, it is proved practically that even with rather basic search strategy and proper control knowledge, planners can achieve surprising effectiveness. We believe that control knowledge can be benefit in improving the performance of heuristic search. In this paper we will show how to combine heuristic search and control knowledge in the same framework.

A promising way of specifying control knowledge is to use temporal logic. For example, the "next" operator from LTL (Linear Temporal Logic) allows one to specify what can and cannot be held at the next time step, while the "always" operator can assert the invariants holding in any state reachable from the initial state. Bacchus et al. develop a planner TLPlan [1], where domain-dependent control knowledge encoded in bounded first-order LTL to prune the search space via standard forward-chaining search. Kvarnström introduces a complex logic, called TAL (Temporal Action Logic), which encodes actions into logical formulas, and also employs this kind of domain-dependent control knowledge in the forward-chaining planner named TALplanner [11]. TLPlan can only generate sequential plans, while TALplanner is capable of producing parallel plans (actions executed concurrently). Baier and McIlraith convert LTL formulas into automata with infinite loops which can be added to a planning task, where additional variable for each state of the automata is introduced in the search space [2]. In their work, LTL is used for expressing the temporally extended goals, not for pruning the search space. Wang et al. present a method to describe landmarks and their associated orderings in LTL [20]. They translate LTL formulas into automata and augment the famous FF heuristic [10] by including the automata variables in the relaxed planning graph, and show such "landmarks" are effective in computing high-quality plans quickly. HTN (Hierarchical Task Network) [6] based planning planners, e.g., SHOP2 [14], in which the dependency among actions is given in advance by hierarchically structured networks. The network represents a hierarchy of tasks each of which can be executed. The task is either primitive, or can be decomposed into refined subtasks. The planning process starts by decomposing the initial task network and continues until all compound tasks are decomposed into primitive ones.

We find the above mentioned work are encountered a challenge in common: it is difficult to solve the planning problem with spatial and temporal constraints, e.g., the CityCar domain in IPC 2014[1], or the DataNetwork domain in IPC 2018[2]. Since the control knowledge formalism is mainly temporal logic or task networks which can only express temporal constraints. Additionally, the current dominant heuristic approaches also solve such problems inefficiently due to

[1] https://helios.hud.ac.uk/scommv/IPC-14/domains_sequential.html.
[2] https://ipc2018-classical.bitbucket.io/#domains.

the lack of useful guiding information. In this paper, in order to enhance the performance of planning systems, we propose an approach based on a forward-chaining heuristic search framework, in which we combine a spatio-temporal logic PPTL$^{\mathrm{SL}}$ for the sake of describing the control knowledge. PPTL$^{\mathrm{SL}}$ uses separation logic [15] as the spatial dimension and PPTL (Propositional Projection Temporal Logic) [4,5,23] as the temporal dimension. Separation logic is a spatial logic for reasoning about heap-manipulating programs. PPTL, an expressive temporal logic [17,18], has already been successfully applied to planning [21], real-time system verification [3], efficient temporal properties verification [22] etc. We illustrate how PPTL$^{\mathrm{SL}}$ formulas can be integrated into the heuristic search in the sequel. In words, this can be done by a progression technique specialized on PPTL$^{\mathrm{SL}}$ formulas.

This paper is organized in the following way: Sect. 2 gives some background about the relevant formalism used in this paper. We explain the novel algorithm combing heuristic search and PPTL$^{\mathrm{SL}}$ in Sect. 3. In Sect. 4, experiments and evaluations are illustrated and analyzed. Finally, we give conclusions and list future work in Sect. 5.

2 Preliminaries

In this section we give the definitions of planning and PPTL$^{\mathrm{SL}}$ formalisms used in this paper.

2.1 Planning Representation

We consider classical planning tasks or planning problems as the following standard definition:

Definition 1 (Planning Task). *A planning task is a tuple* $\mathcal{P} = (\Pi, A, I, G)$, *where*

- Π *is a finite set of propositional variables.*
- A *is a finite set of actions, and for each action* $a \in A$, $a = (pre(a), add(a), del(a), cost(a))$, *where* $pre(a) \subseteq \Pi$, $add(a) \subseteq \Pi$, $del(a) \subseteq \Pi$ *and* $cost(a) \in \mathbb{N}_0$.
- $I \subseteq \Pi$ *is the initial state.*
- $G \subseteq \Pi$ *is the set of goal constraints.*

A state s is defined as a set of propositional variables, i.e., the set of facts, the variables which do not appear in s are assumed to be false (closed world assumption). An action a is applicable to a state s if $pre(a) \subseteq s$. When a is applied to s, the successor state is defined as: $succ_a(s) = (s \backslash del(a)) \cup add(a)$. A sequence of actions $\rho = \langle a_0, a_1, \ldots, a_n \rangle$ is applicable to s_0 if a_0 is applicable to s_0, and a_i is applicable to $succ_{a_{i-1}}(s_{i-1})$, $1 \leq i \leq n$. We abbreviate the notation $succ_{a_n}(\ldots(succ_{a_0}(s))\ldots)$ as $succ_{\langle a_0,\ldots,a_n\rangle}(s)$. When applying a sequence of actions ρ to a state s, a sequence of states from s to $succ_\rho(s)$ is obtained. If $s = I$ and $G \subseteq succ_\rho(s)$, ρ is called a plan. The cost of a plan is obtained as the sum of the cost of each action in the plan, i.e., $\sum_{i=0}^{n} cost(a_i)$.

2.2 Spatio-Temporal Logic PPTL$^{\mathrm{SL}}$

Since there are many planning tasks involving spatial and temporal constraints, we choose to use the spatio-temporal logic PPTL$^{\mathrm{SL}}$ as the control knowledge. Let Var and Loc be the set of spatial variables and spatial locations respectively. nil is employed to represent the inactive location. The syntax of PPTL$^{\mathrm{SL}}$ is defined by the following grammar:

$$
\begin{aligned}
&e ::= nil \mid l \mid x\\
&\phi ::= p \mid GOAL(p) \mid e_1 = e_2 \mid e_0 \mapsto \{\, e_1, \ldots, e_n \,\} \mid \neg\phi \mid \phi_1 \vee \phi_2 \mid \phi_1 \# \phi_2 \mid \exists x : \phi\\
&P ::= \phi \mid \neg P \mid P_1 \vee P_2 \mid \bigcirc P \mid (P_1, \ldots, P_m)\, prj\, P_0 \mid P^*
\end{aligned}
$$

where $p \in \Pi$ is a propositional variable, $x \in Var$ a spatial variable, and $l \in Loc$ a location, ϕ denotes separation logic formulas and P PPTL$^{\mathrm{SL}}$ formulas.

Let $\mathsf{Val} = Loc \cup \{\, nil \,\}$ be the spatial values, and $\mathbb{B} = \{\, true, false \,\}$ the boolean domain. A state s is a triple (I_p, I_v, I_l), where $I_p : \Pi \to \mathbb{B}$, $I_v : Var \to \mathsf{Val}$ and $I_l : Loc \rightharpoonup \bigcup_{i=1}^{n} \mathsf{Val}^i$. Intuitively, in our spatial model, $I_l(l) = (l_1, l_2)$ denotes a spatial cell labeled by l and with (l_1, l_2) stored in. We will refer to the domain of I_l by $dom(I_l)$. We say I_{l_1} and I_{l_2} are disjoint, written as $I_{l_1} \perp I_{l_2}$, if $dom(I_{l_1}) \cap dom(I_{l_2}) = \emptyset$. The operation $I_{l_1} \bullet I_{l_2}$ is defined for the union of I_{l_1} and I_{l_2}. The notation $s[e]$ indicates the evaluation of e with respect to s. Given G as the goal constraints of a planning task, semantics of separation logic formulas is defined by the satisfaction relation $\models_{\mathrm{SL}}$ below.

$\boldsymbol{s \models_{\mathrm{SL}} p}$ iff $p = true$.
$\boldsymbol{s \models_{\mathrm{SL}} e_1 = e_2}$ iff $s[e_1] = s[e_2]$.
$\boldsymbol{s \models_{\mathrm{SL}} e_0 \mapsto \{\, e_1, \ldots, e_n \,\}}$ iff $dom(I_l) = \{s[e_0]\}$ and $I_l(s[e_0]) = (s[e_1], \ldots, s[e_n])$.
$\boldsymbol{s \models_{\mathrm{SL}} \neg\phi}$ iff $s \not\models_{\mathrm{SL}} \phi$.
$\boldsymbol{s \models_{\mathrm{SL}} \phi_1 \vee \phi_2}$ iff $s \models_{\mathrm{SL}} \phi_1$ or $s \models_{\mathrm{SL}} \phi_2$.
$\boldsymbol{s \models_{\mathrm{SL}} \phi_1 \# \phi_2}$ iff $\exists I_{l_1}, I_{l_2} : I_{l_1} \perp I_{l_2}, I_l = I_{l_1} \bullet I_{l_2}, (I_p, I_v, I_{l_1}) \models_{\mathrm{SL}} \phi_1$, and $(I_p, I_v, I_{l_2}) \models_{\mathrm{SL}} \phi_2$.
$\boldsymbol{s \models_{\mathrm{SL}} \exists x : \phi}$ iff $\exists l \in \mathsf{Val} : (I_p, I_v(x/l), I_l) \models_{\mathrm{SL}} \phi$.
$\boldsymbol{s \models_{\mathrm{SL}} GOAL(p)}$ iff $\forall s :$ if G is true in s, then $s \models_{\mathrm{SL}} p$.

The following shows some useful derived formulas. Formula $e \mapsto e_i$ denotes there is a link between e and e_i. $ls^n(e_1, e_2)$ precisely describes a path from e_1 to e_2 of length n without any other locations (e_2 is also excluded). $e_1 \to^+ e_2$ and $e_1 \to^* e_2$ indicate that e_2 is reachable from e_1.

$$
\begin{aligned}
&e \mapsto e_i \stackrel{\mathrm{def}}{=} e \mapsto \{\, e_1, \ldots, e_i, \ldots, e_n \,\}\\
&ls^1(e_1, e_2) \stackrel{\mathrm{def}}{=} e_1 \mapsto e_2 \quad ls^{n+1}(e_1, e_2) \stackrel{\mathrm{def}}{=} \exists x : e_1 \mapsto x \# ls^n(x, e_2)\\
&e_1 \to^+ e_2 \stackrel{\mathrm{def}}{=} \bigvee_{i=1}^{n} ls^i(e_1, e_2) \# true \quad e_1 \to^* e_2 \stackrel{\mathrm{def}}{=} e_1 = e_2 \vee e_1 \to^+ e_2
\end{aligned}
$$

An interval $\sigma = \langle s_0, s_1, \ldots \rangle$ is a nonempty sequence of states, possibly finite or infinite. The length of σ, denoted by $|\sigma|$, is ω if σ is infinite, otherwise it is the number of states minus one. We consider the set N_0 of non-negative integers, define $N_\omega = N_0 \cup \{\omega\}$ and $\preceq$ as $\leq \setminus \{(\omega, \omega)\}$. $\sigma_{(i \ldots j)} (0 \leq i \preceq j \leq |\sigma|)$ denotes the sub-interval $\langle s_i, \ldots, s_j \rangle$. The concatenation of σ with another interval σ' is denoted by $\sigma \cdot \sigma'$. Let $\sigma = \langle s_k, \ldots, s_{|\sigma|} \rangle$ be an interval and $r_1, \ldots, r_n$ be integers such that $0 \leq r_1 \leq \cdots \leq r_n \preceq |\sigma|$. The projection of σ onto $r_1, \ldots, r_n$ is the interval, $\sigma \downarrow (r_1, \ldots, r_n) = \langle s_{t_1}, \ldots, s_{t_m} \rangle$, where $t_1, \ldots, t_m$ are obtained from $r_1, \ldots, r_n$ by deleting all duplicates.

An interpretation of a PPTL$^{\text{SL}}$ formula is a triple $\mathcal{I} = (\sigma, k, j)$ where $\sigma = \langle s_0, s_1, \ldots \rangle$ is an interval, k a non-negative integer and j an integer or ω such that $0 \leq k \preceq j \leq |\sigma|$. We write $(\sigma, k, j) \models P$ to mean that a formula P is interpreted over a sub-interval $\sigma_{(k \ldots j)}$ of σ with the current state being s_k. The precise semantics of PPTL$^{\text{SL}}$ built upon σ is given below.

- $\boldsymbol{\mathcal{I} \models \phi}$ iff $s_k \models_{\text{SL}} \phi$.
- $\boldsymbol{\mathcal{I} \models P_1 \vee P_2}$ iff $\mathcal{I} \models P_1$ or $\mathcal{I} \models P_2$.
- $\boldsymbol{\mathcal{I} \models \neg P}$ iff $\mathcal{I} \not\models P$.
- $\boldsymbol{\mathcal{I} \models \bigcirc P}$ iff $k < j$ and $(\sigma, k+1, j) \models P$.
- $\boldsymbol{\mathcal{I} \models (P_1, \ldots, P_m)\, prj\, P}$ iff $\exists r_0, \ldots, r_m$ and $k = r_0 \leq r_1 \leq \cdots \leq r_m \preceq j$ such that $(\sigma, r_{i-1}, r_i) \models P_i$ for all $1 \leq i \leq m$ and $(\sigma', 0, |\sigma'|) \models P$ for one of the following σ' :
  (a) $r_m < j$ and $\sigma' = \sigma \downarrow (r_0, \ldots, r_m) \cdot \sigma_{(r_m+1 \ldots j)}$
  (b) $r_m = j$ and $\sigma' = \sigma \downarrow (r_0, \ldots, r_{i'})$ for some $0 \leq i' \leq m$.
- $\boldsymbol{\mathcal{I} \models P^*}$ iff $\exists r_0, \ldots, r_n$ and $k = r_0 \leq r_1 \leq \cdots \leq r_{n-1} \preceq r_n = j (n \geq 0)$ such that $(\sigma, r_{i-1}, r_i) \models P$ for all $1 \leq i \leq n$; or $\exists r_0, \ldots,$ and $k = r_0 \leq r_1 \leq r_2 \leq \cdots$ such that $\lim_{i \to \infty} r_i = \omega$ and $(\sigma, r_{i-1}, r_i) \models P$ for all $i \geq 1$.

A formula P is satisfied over an interval σ, written as $\sigma \models P$, if $(\sigma, 0, |\sigma|) \models P$ holds. We also have some derived temporal formulas:

$$\varepsilon \stackrel{\text{def}}{=} \neg \bigcirc true \qquad P_1; P_2 \stackrel{\text{def}}{=} (P_1, P_2)\, prj\, \varepsilon$$

$$\Diamond P \stackrel{\text{def}}{=} true; P \qquad \Box P \stackrel{\text{def}}{=} \neg \Diamond \neg P$$

Here, ε denotes an interval with zero length, $\Box$ and $\Diamond$ have the standard meanings as in LTL. Formula $P_1; P_2$ asserts that P_1 holds from now on until some point in the future, and from that point on, P_2 holds.

Example 1 (Example of Control Knowledge). *The CityCar domain simulates the impact of road building/demolition in traffic networks. Some cars need to move to their destinations, while roads may be built and removed dynamically (the most costly actions). CityCar domain is the second hardest problem in the 8th IPC for the reason that the roads can shape in a variety of ways (most of which are infeasible). Consider the following control knowledge which is useful for the CityCar domain:*

$\Box(\forall j_1, j_2, j_3, j_4 : (\exists r_1, r_2 : road\_connect(r_1, j_1, j_3) \wedge road\_connect(r_2, j_1, j_4)) \rightarrow \neg(j_3 \rightarrow^* j_2 \# j_4 \rightarrow^* j_2))$

The above formula can be read as "for any locations j_1, j_2, j_3, j_4, if there exist two road r_1 and r_2 sharing the same starting location j_1, any two paths between j_1 and j_2 should not be disjoint". $road\_connect(r, j_1, j_2)$ denotes that there is a road r from j_1 to j_2. $\#$ is able to express the disjointness of two paths. Since all variables are bounded, we can use quantifiers to obtain concise formulas. Note that all predicates can be replaced by propositional variables.

Since roads are limited resource, we should keep at most one path from a location to another to reduce the usage of roads. This knowledge provides valuable information for reducing the cost a plan, as well as the search space.

3 Combining PPTL$^{\mathbf{SL}}$ with Heuristic Search

Our aim is to exploit PPTL$^{\mathrm{SL}}$ style control knowledge in heuristic search. We assume control knowledge is given at the beginning of the search in terms of a "global" PPTL$^{\mathrm{SL}}$ formula. We will show how control knowledge can be added into the search process in this section.

3.1 Progression of PPTL$^{\mathbf{SL}}$

We can rewrite PPTL$^{\mathrm{SL}}$ formulas in a special form of what has to be held in the current state and the following states. This can be accomplished by Theorems 1 and 2 [12,13].

Theorem 1 (Equisatisfiable Translation). *For any PPTL^{SL} formula P, there exists an equisatisfiable RPPTL^{SL} formula R which is defined as,*

$$R ::= p \mid GOAL(p) \mid e_1 = e_2 \mid \neg R \mid R_1 \vee R_2 \mid \bigcirc R \mid (R_1, \ldots, R_m)\, prj\, R \mid R^*$$

RPPTL$^{\mathrm{SL}}$ has no formulas describing spatial states. The translation process from PPTL$^{\mathrm{SL}}$ to RPPTL$^{\mathrm{SL}}$ is omitted here for reasons of brevity, please refer to [12,13] for more detail.

Theorem 2 (Normal Form Translation). *Any RPPTL^{SL} formula R can be rewritten into its normal form which is defined as,*

$$R \equiv \bigvee_{j=1}^{m} (\phi_j \wedge \varepsilon) \vee \bigvee_{i=1}^{n} (\phi'_i \wedge \bigcirc R_i)$$

where ϕ_j and ϕ'_i are separation logic formulas, and R_i is a general RPPTL^{SL} formula.

We borrow the basic idea of progression technique from [1]. A RPPTL$^{\mathrm{SL}}$ formula R can be evaluated progressively over a sequence of states $\langle s_0, s_1, \ldots \rangle$. Intuitively, progressing R with $\langle s_0, s_1, \ldots \rangle$ means that we obtain a new formula R' which is satisfied by $\langle s_1, \ldots \rangle$, where the original sequence of states satisfies R. The following definition includes all progression rules required by RPPTL$^{\mathrm{SL}}$.

Definition 2 (Progression Rules). *Let P be a PPTL^{SL} formula, we can first translate P to a RPPTL^{SL} formula R, then rewrite R in its normal form $\bigvee_{j=1}^{m}(\phi_j \wedge \varepsilon) \vee \bigvee_{i=1}^{n}(\phi'_i \wedge \bigcirc R_i)$. R is evaluated over a sequence of states starting with state s. $progress(R, s)$ is recursively defined as follows:*

- $progress(\varepsilon, s) = \varepsilon$
- $progress(\phi, s) = true$ *if* $s \models_{SL} \phi, false$ *otherwise*
- $progress(R_1 \vee R_2, s) = progress(R_1, s) \vee progress(R_2, s)$
- $progress(R_1 \wedge R_2, s) = progress(R_1, s) \wedge progress(R_2, s)$
- $progress(\bigcirc R_1, s) = R_1$

Theorem 3 (Progression). *Given an interval $\sigma = \langle s_0, s_1, \ldots \rangle$, for any RPPTL^{SL} formula R, $progress(R, s_i) \neq \varepsilon$, the following condition must hold:*

$$(\sigma, i+1, |\sigma|) \models progress(R, s_i) \qquad \textit{iff} \qquad (\sigma, i, |\sigma|) \models R$$

Proof. The proof proceeds by induction on R. Since we can translate R into its normal form, there exists the following cases:

- $R \equiv \phi$. If R is a state formula ϕ, $(\sigma, i, |\sigma|) \models R$ if and only if $s_i \models_{\mathrm{SL}} \phi$. By Definition 2, $progress(\phi, s_i) = true$ or $false$ dependents on whether s_i satisfies ϕ. Any state satisfies $true$, falsifies $false$. Hence, this case holds.
- $R \equiv \varepsilon$. If R is ε, $progress(R, s_i) = \varepsilon$ which violates the precondition. Hence, this case holds.
- $R \equiv R_1 \vee R_2$. $(\sigma, i, |\sigma|) \models R$ if and only if $(\sigma, i, |\sigma|) \models R_1$ or $(\sigma, i, |\sigma|) \models R_2$. By inductive hypothesis, $(\sigma, i, |\sigma|) \models R_1$ if and only if $(\sigma, i+1, |\sigma|) \models progress(R_1, s_i)$, and $(\sigma, i, |\sigma|) \models R_2$ if and only if $(\sigma, i+1, |\sigma|) \models progress(R_2, s_i)$. Hence, $(\sigma, i, |\sigma|) \models R$ if and only if $(\sigma, i+1, |\sigma|) \models progress(R_1, s_i) \vee progress(R_2, s_i)$. By Definition 2, $progress(R_1, s_i) \vee progress(R_2, s_i) = progress(R_1 \vee R_2, s_i)$.
- $R \equiv R_1 \wedge R_2$. Similar to the proof of the case $R \equiv R_1 \vee R_2$.
- $R \equiv \bigcirc R_1$. Based on the semantics of $\bigcirc$, $(\sigma, i, |\sigma|) \models R$ if and only if $(\sigma, i+1, |\sigma|) \models R_1$. By Definition 2, $(\sigma, i+1, |\sigma|) \models progress(R, s_i)$. □

As we will see later, the progression technique is the essential when using PPTL$^{\mathrm{SL}}$ in heuristic planning process.

3.2 Incorporation of Heuristic Search Framework with PPTL$^{\mathrm{SL}}$

In fact the planning search space is a tree with the root be the initial state. When exploring the space, heuristic algorithms mainly compute heuristic functions to choose which nodes to expand. The major difference or the key is the functions they use according to different applications. The algorithms usually maintain two queues to save nodes. One is a priority queue called open list in which saves

Algorithm 1. Planning algorithm with heuristic search and PPTL$^{\text{SL}}$ control knowledge

Input: Initial state I, goal G, actions A, PPTL$^{\text{SL}}$ formula P.
Output: *Plan*.

$g(I) = 0$;
$h(I) =$ heuristic estimate for I;
$priority(I) = g(I) + h(I)$;
$open = \{ I \}$;
$closed = \emptyset$;
$Plan = \langle\rangle$;
Translate P_I to an equisatisfiable RPPTL$^{\text{SL}}$ formulas R_I;
$R_I = progress(R, I)$;
while $open \neq \emptyset$ **do**
  select $s \in open$ which has the highest priority;
  $closed = closed \cup \{ s \}$;
  if $G \subseteq s$ **then**
    return *Plan*;
  end if
  for action $a \in A$ applicable to s **do**
    $s' = succ_a(s)$;
    $R_{s'} = progress(R_s, s')$;
    if $R_{s'}$ is unsatisfiable **then**
      continue;
    end if
    $Plan = Plan \cdot a$;
    $g(s') = g(s) + cost(a)$;
    $h(s') =$ heuristic estimate for s';
    if $\nexists s'' : s'' \in open \cup closed$ and $s'' = s'$ **then**
      $open = open \cup \{ s' \}$;
      $priority(s') = g(s') + h(s')$;
    else if $\exists s'' : s'' \in open \cup closed$ and $s'' = s'$ and $g(s') < g(s'')$ **then**
      $priority(s') = g(s') + h(s')$;
      if $s' \in closed$ **then**
        $closed = closed \setminus \{ s' \}$;
        $open = open \cup \{ s' \}$;
      end if
    else
      remove tail a of *Plan*;
    end if
  end for
end while
return no solution;

unexpanded nodes, and the other is called closed list to record expanded ones. The node with the highest priority in the open list will be selected for the future expansion.

For example, the best-first search algorithm A* [7], examines which node n in the open list has the lowest value with respect to $f(n) = g(n) + h(n)$, which means n possesses the highest priorities. $g(n)$ represents the exact cost from the initial node to the current node n, and $h(n)$ represents the heuristic estimated cost from n to the goal. Specifically, $h(n)$ can be calculated in various ways for different purpose. Our algorithm employs A* as the underlying heuristic search framework, since A* is guaranteed to return an optimal plan if the heuristic function h used by A* is admissible. A heuristic function is said to be admissible if it never overestimates the cost of reaching the goal.

The corresponding pseudo code is shown in Algorithm 1. For simplicity, we just use states to represent nodes. The algorithm takes the planning problem and the control formula as input, and searches for a plan. At the beginning of the algorithm, several initializations are done in line 1–8. The exact cost of the initial state $g(I)$ is assigned to 0, the heuristic estimate $h(I)$ is also calculated (depends on implementation). The open list is initialized with a single element I with priority $g(I) + h(I)$, the closed list and the action sequence $Plan$ is empty. The global formula P (control knowledge) is translated into an equisatisfiable RPPTL$^{\text{SL}}$ formula R. Then R with the initial state is progressed, resulting in R_I. For each iteration of the main loop, we first select an element s which has the highest priority from the open list and put it in the closed list. As long as s satisfies the goal, i.e., $G \subseteq s$, a plan ($Plan$) is found and returned (line 12–14). Otherwise we try to execute an action a which is applicable to s, and the successor state s' is obtained. Then the current control knowledge R_s is progressed with s'. If the progression result R'_s is unsatisfiable, node s' will be discarded by trying another applicable action (line 18–20). If not, from line 21 to 23, we add a to $Plan$, calculate the exact cost $g(s')$ for reaching s', and estimate the heuristic cost $h(s')$. After that, the algorithm checks if s' is in the open list or the closed list. There exists three different cases. First, if s' is a new state, it will be inserted into the open list with priority $g(s') + h(s')$. Second, if s' is not a new state, but $g(s')$ is lower than before, we simply updates the g value of s'. Note that we do not need to recalculate the heuristic estimate since it is only state dependent. Moreover, if s' is already in the closed list, we should take it from the closed list and insert it into the open list again. Third, if s' is not new and has a higher g value, we do nothing but only remove the tail of the $Plan$ since the existing g value is at least good or even better than the newly found one. Finally, if the main loop ends without finding a solution, the algorithm will report that the problem is unsolvable. In the algorithm, the progression technique will be helpful to prune more state space by giving additional restrictions together with heuristics.

We often hope to find a plan with the cost as low as possible. To this end, we can slightly modify the algorithm by providing a bound cost initially, thus any plan over cost will not be returned.

4 Experiment

The above mentioned framework has been implemented on top of *S-TSolver* [12,13], a domain-dependent planner using basic search (e.g., depth first search). Note that we do not specify any heuristic strategy in Algorithm 1. In the implementation, a traditional heuristic method is employed via relaxed planning task exploration.

Definition 3 (Relaxed Planning Task). *Given a planning task* $\mathcal{P} = (\Pi, A, I, G)$*, the relaxation* $\mathcal{P}'$ *of* $\mathcal{P}$ *is defined as* $\mathcal{P}' = (\Pi, A', I, G)$*, with*

$$A' = \{\, (pre(a), add(a), \emptyset, cost(a)) \mid (pre(a), add(a), del(a), cost(a)) \in A \,\}$$

In words, one obtains the relaxed planning task by ignoring the delete effects of all actions. The basic idea behind the relaxed planning task is that the number of facts in a state during search is guaranteed to be increased or stable compared with the previous state. Given a planning task $\mathcal{P} = (\Pi, A, I, G)$ and a state s, we estimates the cost of a relaxed plan that achieves the goals starting from s for the task $\mathcal{P}' = (\Pi, A', s, G)$. More concretely, the base heuristic estimates a rough approximation as the following cost values:

$$hcost_s(p) = \begin{cases} 0 & \text{if } p \in s \\ n + cost(a) & \text{if } min\{\, \Sigma_{q \in pre(a)} hcost_s(q)) \mid a \in A', p \in add(a) \,\} = n \\ \infty & \text{otherwise} \end{cases}$$

Given a set of facts, we assume they are achieved independently in the sense that the *hcost* of the facts is estimated as the sum of the individuals. The heuristic estimate for a state is:

$$h(s) = hcost_s(G) = \Sigma_{g \in G} hcost_s(g)$$

The assumptions that each fact is achieved independently ignores positive interactions of actions. In fact, two facts may be achieved exclusively. However, it can be proved that relaxed planing task is solvable in polynomial time.

Example 2 (An Example of Heuristic Estimate). *Consider the following example of a simple planning task, where* $G = \{\, g_1, g_2 \,\}$ *and three actions* $a_1 = (\{\, p \,\}, \{\, g_1 \,\}, \emptyset, 1), a_2 = (\{\, q \,\}, \{\, g_2 \,\}, \emptyset, 1), a_3 = (\{\, p \,\}, \{\, q \,\}, \emptyset, 1)$*. Given two states* $s_1 = \{\, p \,\}, s_2 = \{\, q \,\}$*, the heuristic estimate for* s_1 *and* s_2 *are:*

$$\begin{aligned} h(s_1) &= hcost_{s_1}(G) \\ &= hcost_{s_1}(g_1) + hcost_{s_1}(g_2) \\ &= hcost_{s_1}(p) + cost(a_1) + hcost_{s_1}(q) + cost(a_2) \\ &= 0 + 1 + hcost_{s_1}(p) + cost(a_1) + 1 \\ &= 1 + 2 = 3 \end{aligned}$$

$$\begin{aligned} h(s_2) &= hcost_{s_2}(G) \\ &= hcost_{s_2}(g_1) + hcost_{s_2}(g_2) \\ &= \infty + hcost_{s_2}(q) + cost(a_2) \\ &= \infty + 1 = \infty \end{aligned}$$

Obviously, s_1 has better heuristic value than s_2.

We extend the planner, *S-TSolver*, by replacing the basic forward search with heuristic mechanism. Two standard input files (written in PDDL) namely the domain file and problem file are needed, the former gives the definitions of actions and the latter lists the initial state and the goal constraints. At each step, the heuristic value of every new reachable state (after applying an action) is calculated, while only the states satisfy the control knowledge are maintained.

Experiment has been carried out on *S-TSolver*. We compare *S-TSolver* of two modes, one with basic search and the other with heuristic search. We use a timeout of 10 min per run. To evaluate the plans found by the search, the search time and cost for each plan are considered as metrics. Table 1 shows the result of running *S-TSolver* for the CityCar domain. In the table, the column "cost" indicates the minimum plan cost found by the planner, "time" the search time and "len" the length of the plan. The column *S-TSolver* shows the result with basic search and control knowledge, and *S-TSolver*(heuristic) shows the result with both heuristic search and control knowledge. We do not compare the mode that *S-TSolver* uses only heuristic search without control knowledge. The reason is that the heuristic is so weak that only several solutions can be given for few instances. Even Fast Downward [9], one of the best heuristic planner so far, adopting complex and delicate heuristics only gives solutions for five simple instances[3].

The minimum cost and least time for each instance are in bold. In general, the higher the cost, the longer the length. Heuristic really helps the planner to improve the quality of a plan. In particular, all plans found by *S-TSolver*(heuristic) have better quality than those by *S-TSolver*. However, *S-TSolver* takes the advantage of search time, i.e., mostly the plans' search time is less since *S-TSolver*(heuristic) spends additional time to compute the heuristic estimates. But this is not all the cases, sometimes heuristic enables the search to move closer towards goals as soon as possible. Instead, the search time is not affected (increased) by computing heuristics, especially for some difficult instances (e.g., p17–p19).

[3] https://helios.hud.ac.uk/scommv/IPC-14/resDoc.html.

Table 1. Performance of *S-TSolver* with (without) heuristic: CityCar domain

Problem instance	*S-TSolver*			*S-TSolver*(heuristic)		
	cost	time	len	cost	time	len
p01	**70**	0.50	17	**70**	**0.08**	17
p02	126	**6.03**	27	**122**	57.18	33
p03	106	**0.79**	27	**100**	4.24	32
p04	**136**	**4.92**	41	**136**	11.10	41
p05	94	**1.60**	37	**86**	4.52	39
p06	154	**4.22**	23	**138**	4.34	28
p07	154	**3.69**	23	**134**	23.28	23
p08	**114**	1.98	22	**114**	**1.22**	22
p09	108	**5.77**	28	**102**	18.12	32
p10	176	16.01	38	**152**	**15.10**	34
p11	**184**	8.88	36	**184**	**3.46**	38
p12	202	**5.00**	36	**158**	31.88	40
p13	350	**26.31**	62	**206**	27.02	52
p14	164	**11.63**	48	**126**	13.64	38
p15	260	21.22	48	**239**	**21.12**	47
p16	150	**12.00**	44	**136**	22.24	39
p17	304	33.06	56	**230**	**19.74**	48
p18	180	19.43	56	**144**	**14.20**	49
p19	416	38.29	84	**240**	**1.18**	60
p20	280	**44.14**	82	**204**	110.54	62

5 Conclusion

This paper introduces an algorithm that combines control knowledge and heuristic search in the same framework. PPTL$^{\mathrm{SL}}$ formulas representing control knowledge are evaluated at each planning step by progression technique. Then we incorporate progression of PPTL$^{\mathrm{SL}}$ formulas with a general forward heuristic search. The experiment demonstrates that the effectiveness and efficiency of heuristic search can be further improved by exploiting domain specific knowledge. Generally speaking, our approach belongs to model checking framework. In the future, we will plan to exploit some existing efficient model checking techniques, e.g., abstract model checking [16,19], logic based model checking [24], in our work. We believe that one will obtain better results in this way.

References

1. Bacchus, F., Kabanza, F.: Using temporal logics to express search control knowledge for planning. Artif. Intell. **116**(1–2), 123–191 (2000). https://doi.org/10.1016/S0004-3702(99)00071-5
2. Baier, J.A., McIlraith, S.A.: Planning with first-order temporally extended goals using heuristic search. In: Proceedings of the Twenty-First National Conference on Artificial Intelligence and the Eighteenth Innovative Applications of Artificial Intelligence Conference, July 16–20, 2006, Boston, Massachusetts, USA, pp. 788–795 (2006). http://www.aaai.org/Library/AAAI/2006/aaai06-125.php
3. Cui, J., Duan, Z., Tian, C., Du, H.: A novel approach to modeling and verifying real-time systems for high reliability. IEEE Trans. Reliab. **67**(2), 481–493 (2018). https://doi.org/10.1109/TR.2018.2806349
4. Duan, Z., Tian, C.: A practical decision procedure for propositional projection temporal logic with infinite models. Theor. Comput. Sci. **554**, 169–190 (2014). https://doi.org/10.1016/j.tcs.2014.02.011
5. Duan, Z., Tian, C., Zhang, N.: A canonical form based decision procedure and model checking approach for propositional projection temporal logic. Theor. Comput. Sci. **609**, 544–560 (2016). https://doi.org/10.1016/j.tcs.2015.08.039
6. Erol, K., Hendler, J.A., Nau, D.S.: HTN planning: complexity and expressivity. In: Proceedings of the 12th National Conference on Artificial Intelligence, Seattle, WA, USA, July 31–August 4, 1994, vol. 2, pp. 1123–1128 (1994). http://www.aaai.org/Library/AAAI/1994/aaai94-173.php
7. Hart, P.E., Nilsson, N.J., Raphael, B.: A formal basis for the heuristic determination of minimum cost paths. IEEE Trans. Syst. Sci. Cybern. **4**(2), 100–107 (1968). https://doi.org/10.1109/TSSC.1968.300136
8. Helmert, M.: Complexity results for standard benchmark domains in planning. Artif. Intell. **143**(2), 219–262 (2003). https://doi.org/10.1016/S0004-3702(02)00364-8
9. Helmert, M.: The fast downward planning system. J. Artif. Intell. Res. **26**, 191–246 (2006). https://doi.org/10.1613/jair.1705
10. Hoffmann, J., Nebel, B.: The FF planning system: fast plan generation through heuristic search. J. Artif. Intell. Res. **14**, 253–302 (2001). https://doi.org/10.1613/jair.855
11. Kvarnström, J., Magnusson, M.: Talplanner in IPC-2002: extensions and control rules. CoRR. http://arxiv.org/abs/1106.5266 (2011)
12. Lu, X., Tian, C., Duan, Z.: Temporalising separation logic for planning with search control knowledge. In: Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19–25, 2017, pp. 1167–1173 (2017). https://doi.org/10.24963/ijcai.2017/162
13. Lu, X., Tian, C., Duan, Z., Du, H.: Planning with spatio-temporal search control knowledge. IEEE Trans. Knowl. Data Eng. **30**(10), 1915–1928 (2018). https://doi.org/10.1109/TKDE.2018.2810144
14. Nau, D.S., et al.: SHOP2: an HTN planning system. J. Artif. Intell. Res. (JAIR) **20**, 379–404 (2003). https://doi.org/10.1613/jair.1141
15. Reynolds, J.C.: Separation logic: a logic for shared mutable data structures. In: Proceedings 17th IEEE Symposium on Logic in Computer Science (LICS 2002), July 22–25, 2002, Copenhagen, Denmark, pp. 55–74 (2002). https://doi.org/10.1109/LICS.2002.1029817

16. Tian, C., Duan, Z., Duan, Z., Ong, C.L.: More effective interpolations in software model checking. In: Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering, ASE 2017, Urbana, IL, USA, October 30–November 03, 2017, pp. 183–193 (2017). https://doi.org/10.1109/ASE.2017.8115631
17. Tian, C., Duan, Z.: Complexity of propositional projection temporal logic with star. Math. Struct. Comput. Sci. **19**(1), 73–100 (2009). https://doi.org/10.1017/S096012950800738X
18. Tian, C., Duan, Z.: Expressiveness of propositional projection temporal logic with star. Theor. Comput. Sci. **412**(18), 1729–1744 (2011). https://doi.org/10.1016/j.tcs.2010.12.047
19. Tian, C., Duan, Z., Duan, Z.: Making CEGAR more efficient in software model checking. IEEE Trans. Softw. Eng. **40**(12), 1206–1223 (2014). https://doi.org/10.1109/TSE.2014.2357442
20. Wang, L., Baier, J., McIlraith, S.: Viewing landmarks as temporally extended goals. In: ICAPS 2009 Workshop on Heuristics for Domain-Independent Planning, pp. 49–56 (2009)
21. Yang, K., Duan, Z., Tian, C., Zhang, N.: A compiler for MSVL and its applications. Theor. Comput. Sci. **749**, 2–16 (2018). https://doi.org/10.1016/j.tcs.2017.07.032
22. Yu, B., Duan, Z., Tian, C., Zhang, N.: Verifying temporal properties of programs: a parallel approach. J. Parallel Distrib. Comput. **118**(Part), 89–99 (2018). https://doi.org/10.1016/j.jpdc.2017.09.003
23. Zhang, N., Duan, Z., Tian, C.: A complete axiom system for propositional projection temporal logic with cylinder computation model. Theor. Comput. Sci. **609**, 639–657 (2016). https://doi.org/10.1016/j.tcs.2015.05.007
24. Zhang, N., Duan, Z., Tian, C.: Model checking concurrent systems with MSVL. Sci. China Inf. Sci. **59**(11), 118101 (2016). https://doi.org/10.1007/s11432-015-0882-6