Boost the Integration of Planning and Scheduling: a Heuristics Approach

Yuechang Liu*, Yishan Fang

Jiaying University, Rd. Meisong, Meizhou 514015, China

Abstract

Temporal planning embodies aspects of both planning and scheduling. Many temporal planners handle these two sub-problems in a loose coupling way. This way simplifies the temporal planning problem but restricts the modeling power. In particular, the simplification fails to handle such temporal planning problems that require concurrency, where actions must execute concurrently to achieve expected effects. For those temporally expressive planning problems, the problem of how to integrate planning with scheduling is emphasized for the sake of both finding a valid plan and further, in an effective way. This paper examines three factors that affect the integrated system’s efficiency: information sharing, computation burden balance and interaction frequency. The approach attributes to designing a set of heuristics. By conducting preliminary experiments, the results show good performance of those heuristics compared with the start-of-the-art planner VHPOP.

© 2011 Published by Elsevier Ltd. Selection and/or peer-review under responsibility of Harbin University of Science and Technology Open access under CC BY-NC-ND license.

Keywords: AI planning, scheduling, integration, temporal reasoning

1. Introduction

In classical planning, time is mainly concerned with its qualitative aspect, which is reflected by the ordering between actions. Practical requirement of explicitly representing and reasoning about time in planning led to the emergence of temporal planning. There have been several temporal planning languages varying in expressive power. Specifically for PDDL 2.1 [1] Cushing, Kambhampati, Mausam and Weld made some theoretical analysis and classified them as temporally simple and temporally expressive [2]. Furthermore, they identified temporal planning problems with required concurrency.

* Corresponding author. Tel.: 18688535015;
E-mail address: ycliu@jyu.edu.cn.
These planning problems pose the need of integrating a scheduling module in planning to gain decent solution efficiency.

To bridge the gap between planning and scheduling, Frank and Jonsson proposed a new temporal language - Constraint Based Interval (CBI) planning [3], in which temporal and resource reasoning are equally important to action reasoning. Since then the integration of planning and scheduling is intensively studied by planning researchers. Cesta et. al. and Briel et. al. studied different approaches to loosely couple planning and scheduling [4, 5]. Based on CSP Garrido and Eva Onaindía studied the integration via conflict resolution in integrated system [6], and proposed a flexible architecture to tackle the integration problem [7]. Andrew Coles et. al. studied the spectrum between planning and scheduling [8]. In this paper, a temporal language called sub-CBI planning language is considered. Three factors that are possible to affect integrated system’s efficiency are investigated:

- Computation burden balance. Simple Temporal Problem (STP) [9] is used for the temporal model by many temporal planners. In this search, Disjunctive Temporal Problem (DTP) [10] is adopted, which has the advantage of balancing computation between planning and scheduling [11].

- Memory share: A memory called distance graph (DG) is used for sharing between planning and scheduling module, which makes the two modules be coupled tightly, but interact more efficiently.

- Interaction frequency: The calling of scheduling module when planning is quite expensive. So flexible or even stepping calling is more beneficial for the improvement of solving efficiency.

The paper is organized as follows: firstly, the sub-CBI planning formalization is introduced; then, the sub-CBI planning is described. At last preliminary experimental results are displayed.

2. Sub-CBI planning formalization

Definition 1 (Disjunctive Temporal Problem [10]) A disjunctive temporal problem (DTP) defined on variable set \(\{x_1, x_2, ..., x_n\}\) is a collection of disjunctive temporal constraints, i.e. logical disjunctions of inequality relations of the form \(x_i - x_j \leq c\) where \(x_i, x_j\) are variables and \(c\) a constant of integer.

Definition 2 (Sub-CBI action) A Sub-CBI action \(sca\) is a tuple \(sca(cons, effs, D)\) where: \(cons\) and \(effs\) are set of propositions each of which has two build-in variables \(ts\) and \(te\) labeled with a time interval (called TQP - temporally qualified proposition) of the form \(p[ts, te]\), and \(D\) is a set of disjunctive temporal constraints. Moreover, every sub-CBI action \(sca\) has three built-in variables \(dur(sca)\), \(ts(sca)\) and \(te(sca)\) accounting for its duration time, starting and ending instant respectively. Of course, the equation constraint \(te(sca)=ts(sca)+dur(sca)\) always holds.

Definition 3 (Sub-CBI planning) A sub-CBI planning domain \(\Sigma\) is a tuple \(<Q, ACT>\) where \(Q\) is a set of logical propositions that are used for description of system states and \(ACT\) a set of all of the sub-CBI actions. A sub-CBI planning problem \(P\) is a tuple \(<\Sigma, INIT, GOAL>\) that composed of a sub-CBI planning domain, sets of TQPs for initially known facts and goal facts respectively.

3. Global Sub-CBI algorithm

3.1. Sub-CBI planning algorithm

Definition 4 (unsafe TQP) Given a tuple \(<TQPs, D>\), a TQP is unsafe if and only if following condition holds: \(\neg\exists q(t_1, t_2)\in TQPs\ s.t. \ t_1 \leq t_i \wedge t_i \leq t_2\) or \(\exists q(t_1, t_2)\in TQPs\ s.t. \ t_1 \leq t_i \vee t_i \leq t_2\)

The first branch in above condition is the open condition in POCL planning, and the other is threat.

Definition 5 (partial plan) Given a sub-CBI planning problem \(P\), a partial plan is a tuple \(<CL, ALL_TQP, DC, UTQP>\) where \(CL\) is set of causal links, \(ALL_TQP\) set of TQPs that have been achieved as effects by the actions in \(CL\), \(DC\) set of disjunctive temporal constraints, and \(UTQP\) set of unsafe TQPs.
**Definition 6 (plan)** A plan for a planning problem $P$ is a partial plan $<CL, ALL\_TQP, DC, UTQP>$ where $UTQP \neq \emptyset$ and $DC$ is consistent.

**Definition 7 (Resolver)** A resolver for partial plan $<CL, ALL\_TQP, DC, UTQP>$ and unsafe TQP $p(t_s, t_e)$ is a tuple $<a, t_s \leq t \leq t_e>$ (for open conditions) where $a$ is an action, $p(t_s, t_e) \in effs(a)$, or tuple $<\text{NIL}, t_s \leq t_1, t_2 \leq t_e>$ (for threats) where $\neg q[t_1, t_2] \in ALL\_TQP$.

```plaintext
algorithm Sub-CBI
  Input: partial plan $pl=<CL, ALL\_TQP, DC, UTQP>$;
  Output: a solution plan if solvable, $\emptyset$ otherwise;
  1. if $UTQP = \emptyset$ return $<CL, ALL\_TQP, DC, UTQP>$;
  2. $n=flaw\_select(UTQP, DC)$;
  3. for each resolver in RESOLVERS(n)
    4. $c=resolver\_select(RESOLVERS(n), n, DC)$;
  5. $Cons=true$;
  6. if Call_cond() then
    7. $Cons=update\_Constraints(pl, n, c, DC)$;
  8. $res=Sub\_CBI(<CL, ALL\_TQP, DC, UTQP>)$;
  9. if res!=|$\emptyset$ then return res;
  10. recoverState();
  11. return $\emptyset$;
Fig.1. Sub-CBI planning algorithm
```

High-level sub-CBI algorithm is described in Fig. 1. The sub-CBI planning algorithm first chooses a flaw (at line 2), tries to find a resolver for it (line 4). The interaction between the algorithm and DTP solver is at line 6 and 7. Traditionally, the scheduling module is called once a flaw is resolved. Consider that if a selected resolver is not temporally consistent, the consistency will eventually be discovered and cause backtracking. Whether to call consistency checking is decided by Call_cond() procedure. If Call_cond() returns true, the resolver is applied to update the distance graph with updated partial plan (line7). Then, the procedure enters next-round recursion.

### 3.2. Topology based variable ordering heuristics for flaw and resolver selection

Topology based variable ordering (TVO) heuristics for DTP reasoning [12] can be extended for flaw and resolver selection in sub-CBI planning. The heuristics is based on the evaluation of conflict index of simple temporal constraints. For simple temporal constraint $x_i-x_j \leq c$, its conflict index is defined as:

\[
ci(x_i-x_j \leq a) = DG[x_i, x_j]-c
\] (1)

In equation (1), $DG[x_i, x_j]$ is the shortest path length between $x_i$ and $x_j$ in distance graph. Based on the conflict index of simple temporal constraint, the definition of conflict index for disjunctive temporal constraints follows by maximum, sum, average, or power-average estimation [13].

#### (1) Resolver ordering

The flaws and resolvers can be evaluated based on above estimations. For partial plan $p=<CL, ALL\_TQP, DC, UTQP>$ and unsafe TQP $p(t_s, t_e)$,

A. If $p(t_s, t_e)$ is an open condition, the conflict index is defined in equation (2):

\[
ci(r = <a, t_s - t \leq 0 \land t_e - t \leq 0>)
= \begin{cases} 
  ci(t_1 - t_s \leq 0) + ci(t_e - t_s \leq 0), & \text{if } a \in CL \\
  ci(t_1 - t_s \leq 0) + ci(t_e - t_s \leq 0) + \sum \{p[t_u, t_v]\} & \forall p[t_u, t_v] \in cons(a), \text{otherwise}
\end{cases}
\] (2)
where a is an action and \( p(t_1, t_2) \in \text{effs}(a) \). In the equation (2), we have two branches for two possibilities. One is for the actions in \( \text{CL} \), the other is not. According to the "success-first" principle for value ordering in CSP [13], the resolver to be selected should be one with the minimum conflict index.

B. If \( p(t_0, t_e) \) is a threat, it has resolvers \( \{ r_1, r_2, ..., r_l \} \). Then the estimation of the resolvers is:

\[
\text{ci}(<\text{NIL}, d_i>) = \{ \text{ci}(t_e \leq t_{i1} \lor t_{i2} \leq t_e) \mid i = 1, 2, ..., h \}
\]

(1)

2. Flaw ordering

For a flow \( f \) with resolvers \( \{ r_1, r_2, ..., r_l \} \), the evaluation of \( f \) can be defined:

R1. Max estimation:

\[
\text{ci}(f) = \max\{\text{ci}(r_i) \mid i = 1, 2, ..., l\}
\]

(2)

R2. Sum estimation:

\[
\text{ci}(f) = \sum \{\text{ci}(r_i) \mid i = 1, 2, ..., l\}
\]

(3)

R3. Average estimation:

\[
\text{ci}(f) = \frac{\sum \{\text{ci}(r_i) \mid i = 1, 2, ..., l\}}{l}
\]

(4)

Contrast to value ordering, the variable with the maximum conflict index should be prior selected.

3.3. Planning and scheduling interaction

It is not necessary to call scheduling module in every planning round for at least two reasons: first, for some cases when the flaw-resolving operation is guaranteed not to cause any inconsistency. Second, even in some cases a flaw-resolving operation results in inconsistency, it can eventually be detected afterwards. In the sub-CBI algorithm, a function \( \text{Call}_\text{cond}() \) is designed to decide whether to call scheduling module in the iteration. The definition of \( \text{Call}_\text{cond}() \) is open. Using this condition, the calling of scheduling only happens when it returns true (see Fig.2 for step-by-step and Fig. 3 Call-cond() interleaving way.).

4. Preliminary experimental results and conclusions

The preliminary experiment is conducted upon the data set of tempo-depth temporal planning domain and problem series. The algorithm is implemented using heuristics H5 and R1 with the stepping-style interleaving method (step = 1, 10, 50, 100), running on Red Hat with 1.8G HZ CPU and 512MB memory. The experiment results are compared with VHPOP [14], a start-of-the-art POCL planner (Tab.1).

The data reveals that the system runs faster than VHPOP on the experimented problems. Furthermore, it also reveals that stepping interaction between planning and scheduling has good efficiency gain for these problems (also in linearly decreasing to the number of steps).

Acknowledgements

The research is supported by research foundation of Jiaying University under Grant No. 2010KJ206.
References


