Minimum-Time Cooperative Task Planning in Temporal Logic Specifications

Zesen Liu, Meng Guo, Zongkui Li

***Abstract*—We consider the problem of optimally execution time in the tasks of global Linear Temporal Logic(LTL) specifications with cooperative actions.**

***Index Terms*—IEEE, IEEEtran, journal, LaTeX, paper, template.**

I. Introduction

The computation complexity of LTL task planning for Multi-agents system is a problem that always limits application. So there are many efforts attempt to simplify computational process and most existed work focuses on the decomposing of a global specification to sub-task assigned to single agent. One of the technique called *Partial Order Reduction* is to reduce the number of possible orderings and get a partially ordered set. The main purpose is to reduce the state space of the transition system and there are some difference between Computer Science and Multi-agent system technically. The former use it to find a reduced system consists of a single path while the latter want to find all the Partial Order relationship.

II. PRELIMINARIES

In the following, we provide a brief introduction of the concepts that form the basis for our presented work.

A. Linear Temporal Logic

Linear Temporal Logic is a kind of formula language consisted of a set of atomic propositions $\mathcal{AP}$, the Boolean operators, conjunction $\wedge$, disjunction $\vee$ and negation $\neg$, and temporal operators, next $\bigcirc$, until $\mathcal{U}$, always $\Box$, eventually $\Diamond$. LTL formulas over the set $\mathcal{AP}$ of atomic proposition are formed according to the following grammar:

$$\Phi ::= \top | a | \varphi_1 \wedge \varphi_2 | \varphi_1 \vee \varphi_2 | \neg\varphi | \bigcirc \varphi | \varphi_1 \mathcal{U} \varphi_2 | \Box\varphi_2 | \Diamond\varphi_2$$

where $a \in \mathcal{AP}$ and $\top$ means true.

B. Nondeterministic Büchi Automaton

Given an LTL formula Φ, *a Büchi Automaton* $\mathcal{B}$ constructed as

$$\mathcal{B} = (Q, Q_0, \Sigma, \zeta, Q^F)$$

where Q is a set of states; $Q_0 \subseteq Q$ is a set of initial states; Σ is an input alphabet; $\zeta : Q \times \Sigma \rightarrow 2^Q$ is a transition relation; Q^F is a set of accepting states. A sequence of $\sigma = \sigma(1)\sigma(2)\cdots, \sigma(i) \in \Sigma$ is called an input *word*. And a sequence $l = q_1 q_2 q_3 \cdots, q_1 \in Q_0$ is called a *run*. A *run* l is called feasible if all the transition relations are satisfied along the *run* $\sigma(t) \models \zeta(l(t), l(t+1))$ for all t where $\models$ denotes Boolean satisfaction. A run l is accepting if it is feasible and ends in an accepting state $q_n \in Q^F$. More deeply, an *accepting* path can be written as the structure Prefix-suffix such that the prefix part starts from an initial state and ends in an accepting state which executed only once and the suffix part is a circle around an accepting, which executed infinitely often.

Definition 1 (Essential Sequence)**.** *Given a* word $\sigma = \sigma(1)\sigma(2)\cdots$*, it is called* essential *if and only if it describes a run* l *in* $\mathcal{B}$ *and* $\sigma(i)\backslash\{\pi\} \not\models \zeta(l(i), l(i+1))$ *for all* $\pi \in \sigma(i)$*, i.e.* σ *contains only the required propositions.*

To be more intuitive, we can define a graph $\mathcal{G} = (\nu, \varepsilon)$ to describe the $\mathcal{B}$ that ν is the vertices set and $\nu = Q$, ε is the edges set and $\varepsilon : \nu \times \nu.\varepsilon$ stands for transition ζ that $\forall\epsilon = (v_1, v_2) \in \varepsilon$ if and only if $\zeta(v_1, v_2)$ exists and we also defined the essential label function that for an edge $\epsilon = (v_1, v_2)$,$\zeta(\epsilon) = \zeta(v_1, v_2)$. So that $\mathcal{G}$ is a digraph and all *path* begin from the initial vertices set is feasible and the feasible *path* ended in the accepting states Q^F is accepting *path*. Given an accepting *path* $l_p = v_1 v_2 v_3 \cdots$, we can get the essential sequence $l_w = \zeta(v_1, v_2)\zeta(v_2, v_3)\cdots$ as the input symbols list of l_p.

III. Problem Formulation

The problem will be discuss later with the following structure.

A. Multi-agent system

B. Task specification

C. problem definition

IV. Multi-agent Task Planning

Here we discuss how to plan a LTL fomula in Multi-agent system.

A. Pruning the NBA

Given a LTL fomula Φ, we can generate an NBA $\mathcal{B}$ with tools such as *ltl2ba* and store it in a digraph $\mathcal{G}$. The complexity of $\mathcal{G}$ always grows exponentially with the number of nodes and edges. We focus to dispose the edges which have definite physical meaning that map to symbols which stand for tasks in Multi-agent system. To prune the potentially unnecessary

edges in $\mathcal{G}$, we firstly define the redundant edge triangle property as follows:

Definition 2 (Decomposable Edge property)**.** *Given three different nodes v_1, v_2, v_3 in the NBA $\mathcal{G} = (\nu, \varepsilon)$, and the edges $\epsilon_{12} = (v_1, v_2)$,$\epsilon_{23} = (v_2, v_3)$,$\epsilon_{13} = (v_1, v_3)$ exist. We say that the edge ϵ_{13} is decomposable edge if:*

$$\zeta(\epsilon_{12}) \wedge \zeta(\epsilon_{12}) = \zeta(\epsilon_{13})$$

Lemma 1. *If an edge in $\mathcal{G}$ is a decomposable edge, it can be removed from $\mathcal{G}$ without influence the feasibility of the entire task.*

Proof. For an edge $\epsilon = (v_1, v_3)$ in NBA $\mathcal{G}$ satisfied the DE property, this means the task $\sigma = \zeta(\epsilon)$ can be divided into two subtasks σ_1, σ_2 and there exist a node v_2 that $\epsilon_1 = (v_1, v_3)$,$\epsilon_2 = (v_2, v_3)$ with the symbols $\sigma_1 = \zeta(\epsilon_1)$, $\sigma_2 = \zeta(\epsilon_2)$. For any TS that can provide an accepting *word* $\rho = \cdots \varsigma \cdots$ in $\mathcal{G}$ and go through the edge ϵ, it can execute a symbol $\varsigma \models \sigma$ for edge ϵ. Due to $\sigma = \sigma_1 \wedge \sigma_2$, $\varsigma \models \sigma_1 \wedge \sigma_2 \Rightarrow \varsigma \models \sigma_1$,$\varsigma \models \sigma_2$. The temporal ordered in LTL is *time-abstract* and discrete which don't care about duration of each temporal-step. That means TS can provide a derivate *word* $\rho' = \cdots \varsigma\varsigma \cdots$ by splitting the time step with symbols ς. Facing the $\mathcal{G}'$ with ϵ been removed, the first $\varsigma \models \sigma_1$ and the second $\varsigma \models \sigma_2$. So the feasibility of $\mathcal{G}'$ will not be affected. ■

Using DE property to remove all the decomposable edges in $\mathcal{G}$, we can get a pruned NBA $\mathcal{G}^-$ with only basics subtask which greatly reduce the complexity. It is worth mentioning that a decomposable edge can be devided into two decomposable edges and it also can be one sub-edge of another decomposable edge. That means before we find out all decomposable edges in $\mathcal{G}$, we should not prune anyone to avoid the situations that a sub-edge of decomposable edge is removed and this decomposable edge was regard as an undecomposable edge. What's more, the connectivity of $\mathcal{G}^-$ stays the same as that of $\mathcal{G}$. Because for any three nodes v_1, v_2, v_3, removing the decomposable edge will not change connectivity between them. Thus, we do not need to consider and dispose the nodes in $\mathcal{G}^-$ and just inherit the nodes set ν in $\mathcal{G}$.

B. Computing the temporal order of path

In this section, we take a discussion about the dependence relation of each symbol σ and put forward an anytime algorithm in $\mathcal{G}^-$ to find the dependence relations from accepting path. For simplicity, we mainly discuss the prefix paths from initial states to the accepting states and the suffix paths can be solved by this algorithm with minor changes.

The dependence relation is a concept between two symbols σ_1 and σ_2,$\sigma_1 \neq \sigma_2$ in $\mathcal{G}$ to describe whether they need to execute in a certain temporal order or just in a stochastic order.

Definition 3 (Global Dependence Relation)**.** *Given a BA $\mathcal{B} = (Q, Q_0, \Sigma, \zeta, Q^F)$ with $\alpha, \beta \in \Sigma$,$\alpha \neq \beta$. $\Sigma(q)$ is a symbols set that $\Sigma(q) = \{\sigma \in \Sigma | \exists q', \sigma = \zeta(q, q')\}, \alpha = \zeta(q, \alpha(q))$*

1. α and β are independent *(in $\mathcal{B}$) if for any $q \in Q$ with $\alpha, \beta \in \Sigma(q)$:*

$\beta \in \Sigma(\alpha(q))$ and $\alpha \in \Sigma(\beta(q))$ and $\alpha(\beta(q)) = \beta(\alpha(q))$.

2. α and β are dependent *(in $\mathcal{B}$) if α and β are not* independent *in $\mathcal{B}$.*

A pair of independent symbols in a sequence can be carried out in any order without destroy *accepting* character, which can reduce the order constraint during the executive process. However, the definition is strict and sufficient that the relation need satisfied in every state of $\mathcal{B}$ which can be relaxed. In the simplest case, given an accepting path $l = \cdots v_i v_{i+1} v_{i+2} \cdots$, $\alpha = \zeta(v_i, v_{i+1})$,$\beta = \zeta(v_{i+1}, v_{i+2})$, α, β can be executed in any order without destroy the accepting character if $\alpha(\beta(v_i)) = \beta(\alpha(vi)) = v_{i+2}$. That means we don't need to check the *dependence* relation in all states of $\mathcal{B}$. However, it may cause misunderstanding saying "α and β are independent" when S has multiple symbols equal to α or β. So before defining the local dependence relation, we firstly define the *anchor function*. For convenience, given a path $l = v_0 v_1 v_2 \cdots$, we use $S = \sigma_1 \sigma_2 \sigma_3 \cdots, \sigma_i = \zeta(v_i, v_{i+1})$ as the essential sequence of l and we call the S an *accepting sequence* if l is an accepting path.

Definition 4 (Anchor function)**.** *Given an accepting sequence $S = \sigma_1 \sigma_2 \sigma_3 \cdots \sigma_n$, we defined an anchor function $\Omega : N \to \sigma$ to map the symbols from serial number as $\Omega(i) = \sigma_i$ and $\Omega([1:n]) = S$. $S_\Omega = S$ is the anchor sequence of Ω.*

Definition 5 (Local Dependence Relation)**.** *Given an anchor function $\Omega(i) = \sigma_i$ of an accepting sequence S, $\mathcal{N}$ is a permutation of the natural numbers $[1:n]$ and $\Omega(\mathcal{N})$ is an accepting sequence. $i \in [1 : n-1]$ and $\mathcal{N}'$ is a permutation with $\mathcal{N}'([1:i-1]) = \mathcal{N}([1:i-1])$,$\mathcal{N}'([i+2:n]) = \mathcal{N}([i+2:n])$, $\mathcal{N}'(i) = \mathcal{N}(i+1)$, $\mathcal{N}'(i+1) = \mathcal{N}(i)$.*
If $\Omega(\mathcal{N}(i)) \neq \Omega(\mathcal{N}(i+1))$:
1. $\mathcal{N}(i)$ and $\mathcal{N}(i+1)$ are independent *in $\mathcal{N}$ from Ω if $\Omega(\mathcal{N}')$ is also an accepting sequence.*
2. $\mathcal{N}(i)$ and $\mathcal{N}(i+1)$ are dependent *in $\mathcal{N}$ from Ω if $\Omega(\mathcal{N}')$ is not an accepting sequence.*
If $\Omega(\mathcal{N}(i)) = \Omega(\mathcal{N}(i+1))$:
$\mathcal{N}(i)$ and $\mathcal{N}(i+1)$ are dependent *in $\mathcal{N}$ from Ω*

Definition 6 (Partially ordered set)**.** *A Partially ordered set is a tuple:*

$$P = (\Omega, C_{parallel}, C_{partial}, \mathcal{S}_{accept})$$

where Ω is the anchor function*; $C_{parallel}$ is a set of natural Numbers pair (i, j) which satisfied $\Omega(i)$ and $\Omega(j)$ are* independent*; $C_{partial}$ is a set of natural Numbers pair (i, j) which satisfied $\Omega(i)$ and $\Omega(j)$ are* dependent*; $\mathcal{S}_{accept}$ is the arrangement of nature numbers set whose arrange satisfied the order of $S_{less-than}$. What's more, the anchor sequence $S_\Omega \in \mathcal{S}_{accept}$*

Remark 1. *The chose of anchor function is insignificant, any sequence S in $\mathcal{S}_{accept}$ can use as a new anchor sequence $S'_\Omega = S$ and generate a new anchor function Ω'. And we can find another Partially ordered set P' with same $S'_{accept} = S_{accept}$.*

Under definition above, We can propose algorithm 1 to generate the partial order set of anchor sequence S_Ω in NBA $\mathcal{B}$.

Algorithm 1 P_O_S

Input: The NBA $\mathcal{B}$, a feasible sequence S
Output: Partial order sets P

Set the anchor function Ω with S
$\mathcal{N}_{new} = \{[1:n]\}$# n is the length of S
while $\mathcal{N}_{new} \neq \Phi$ **do**
  Fetch one sequence $\hat{N}$ from $\mathcal{N}_{new}$
  add $\Omega(\hat{N})$ to $\mathcal{S}_{accept}$
  for i in $1:n-1$ **do**
    $\hat{N}' = \hat{N}$, $\hat{N}'(i) = \hat{N}(i+1)$, $\hat{N}'(i+1) = \hat{N}(i)$
    if $\Omega(\hat{N}')$ is accepting in $\mathcal{B}$ & $\Omega(\hat{N}'(i)) \neq \Omega(\hat{N}'(i+1))$ **then**
      add $(\hat{N}'(i), \hat{N}(i+1))$ to $C_{parallel}$
      add $\hat{N}'$ to $\mathcal{N}_{new}$
    else add $(N'(i), N(i+1))$ to $C_{partial}$
  remove $\hat{N}$ from $\mathcal{N}_{new}$
Return $P = (\Omega, C_{parallel}, C_{partial}, \mathcal{S}_{accept})$

Due to the complexity of NBA $\mathcal{B}$, we propose an anytime *algorithm.2*, which can quickly generate a not exact solutions and then after a couple of times gradually improve the quality of the solution, to get the Partial order sets S. This algorithm can be thought of as a generalization of Depth-first search. We can stop the for loop and check the sets of Partial order sets P.

Algorithm 2 anytime algorithm

Input: The NBA $\mathcal{B}$
Output: set of Partial order set $\mathcal{P} = \{P_i\}$

for q_0 in Q_0 **do** # prefix
  $stack = [q_0]$,$S_{visit} = []$,10
  while $stack \neq \Phi$ **do**
    fetch the last one of $stack$ as l_s and remove it
    Find the successors of $l_s(-1)$ as $nodes$
    for $node$ in $nodes$ **do**
      if $node$ not in l_s **then**
        $\hat{l}_s = [l_s, node]$,add $\hat{l}_s$ to $stack$
        if $\hat{l}_s$ is accepting & $\hat{l}_s \notin S_{visit}$ **then**
          $P = P\_O\_S(\mathcal{B}, \hat{l}_s)$,add P to $\mathcal{P}$
          add $\mathcal{S}_{accept}$ to S_{visit}
Return $\mathcal{S}$

C. Task assignment

D. Task Execute

APPENDIX A
A.TRIMMING THE NBA

Appendix one text goes here.

APPENDIX B

Appendix two text goes here.

ACKNOWLEDGMENT

The authors would like to thank...

REFERENCES

[1] H. Kopka and P. W. Daly, *A Guide to LaTeX*, 3rd ed. Harlow, England: Addison-Wesley, 1999.

PLACE PHOTO HERE

Michael Shell Biography text here.

John Doe Biography text here.

Jane Doe Biography text here.

B. Find all feasible path

4) Go to step 3 until the path list get null.
5) Return the path list L_p.

Lemma 2. *1234*