MT* : Multi-Robot Motion Planning with Temporal Goals

Submission # 28

Abstract

We address the problem of generating a motion plan for a team of robots satisfying a complex high-level mission specification given in the form of LTL formulae. The state-of-the-art approach to this problem employs the automata-theoretic model checking technique to solve this problem. This approach involves computation of a product graph of the Büchi automaton generated from the LTL specification and a joint transition system which captures the collective motion of the robots and then computation of the shortest path using Dijkstra's shortest path algorithm to generate the motion plan. We propose MT*, an algorithm that reduces the computation burden for generating such motion plans for multi-robot systems significantly. Our approach generates a reduced version of the product graph without computing the complete joint transition system, which is computationally expensive. It then divides the complete mission specification among the participating robots and generates the trajectories for the individual robots independently. Our approach demonstrates substantial speedup in terms of computation time over the state-of-the-art approach, and unlike the state of the art approach, scales well with both the number of robots and the size of the workspace.

1 Introduction

Motion planning is one of the core problems in robotics where we design algorithms to enable autonomous robots to carry out a real-world complex task successfully (LaValle 2006). A basic motion planning task consists of point-to-point navigation while avoiding obstacles and satisfying some user-given constraints. To solve this problem, there exist many methods among which graph search algorithms like A* (Hart, Nilsson, and Raphael 1968) and sampling based motion planning techniques such as rapidly exploring random trees (LaValle and James J. Kuffner 2001) are two prominent ones. Recently, there has been an increased interest in specifying complex motion plans using temporal logic (e.g. (Kress-Gazit, Fainekos, and Pappas 2007; Karaman and Frazzoli 2009; Bhatia, Kavraki, and Vardi 2010b; Wongpiromsarn, Topcu, and Murray 2012; Chen, Tůmová, and Belta 2012; Ulusoy et al. 2013; Saha et al. 2014)). Using temporal logic (Baier and Katoen 2008), one can specify requirements that involve temporal relationship between different operations performed by robots. Such requirements arise in many robotic applications, including persistent surveillance (Ulusoy et al. 2013), assembly planning (Halperin, Latombe, and Wilson 1998), evacuation (Rodríguez and Amato 2010), search and rescue (Jennings, Whelan, and Evans 1997), localization (Fox et al. 2000), object transportation (Rus, Donald, and Jennings 1995), and formation control (Balch and Arkin 1998).

A number of algorithms exist for solving Linear Temporal logic (LTL) motion planning problems in different settings (e.g (Bhatia, Kavraki, and Vardi 2010a), (Kantaros and Zavlanos 2017), (Vasile and Belta 2013), (Smith et al. 2010), (Ulusoy et al. 2013)). For an exhaustive review on this topics, the readers are directed to the survey by Plaku and Karaman (Plaku and Karaman 2016). In this paper, we focus on the class of multi-robot LTL motion planning problems where the robots have *discrete dynamics* and seek to design a computationally efficient algorithm to generate an optimal trajectories for the robots. Traditionally, the LTL motion planning problem for the robots with discrete dynamics is reduced to the problem of finding the shortest path in a weighted graph and Dijkstra's shortest path algorithm (Cormen et al. 2009) is employed to generate an optimal trajectory satisfying an LTL query (Smith et al. 2010; Ulusoy et al. 2013). However, for a large workspaces and a complex LTL specification, this approach is merely scalable.

Heuristics based search algorithms such as A* (Russell and Norvig 2009) have been successfully used in solving point-to-point motion planning problems and is proven to be significantly faster than Dijkstra's shortest path algorithm. A* algorithm has not been applied to temporal logic path planning as there is no notion of a single destination state in LTL motion planning. In this paper, we introduce the MT* algorithm that for the first time attempts to incorporate the A* search in LTL path planning to generate an *optimal* trajectory for a multi-robot system satisfying an LTL query efficiently. We apply our algorithm to solving various LTL motion planning problems for a multi-robot system in 2-D workspaces and compared the results with that of the algorithm presented in (Ulusoy et al. 2013). Our experimental results demonstrate that MT* in many cases achieves an order of magnitude better computation time than that of the traditional approach (Ulusoy et al. 2013) to solve LTL motion planning problems.

2 Preliminaries

2.1 Workspace, Robot Actions and Trajectory

In this paper, we assume that a team of n robots operates in a 2-D or a 3-D discrete workspace $\mathcal{W}$ which we represent as a grid map. The grid divides the workspace into square shaped cells. Every cell in the workspace $\mathcal{W}$ can be referenced using its coordinates. Some cells in the grid can be marked as obstacles and cannot be visited by any robot. We denote set of obstacles using $\mathcal{O}$.

We capture the motion of a robot using a set of actions Act. The robot changes its state in the workspace by performing the actions from Act. An action $act \in Act$ is associated with a *cost* which captures the energy consumption or time delay (based on the need) to execute it. A robot can move to satisfy a given specification by executing a sequence of actions in Act generating a *trajectory* of states it attains. The *cost of a trajectory* is the sum of costs of the actions to generate the trajectory.

2.2 Transition System

We model the motion of the robot i in the workspace $\mathcal{W}$ as a weighted transition system defined as

$$T^i := (S^i, s_0^i, E^i, \Pi^i, L^i, w^i)$$

where (i) S^i is the set of states/vertices, (ii) $s_0^i \in S^i$ is the initial state of the robot i, (iii) $E^i \subseteq S^i \times S^i$ is the set of transitions/edges allowed to be taken by robot i, $(s_1^i, s_2^i) \in E^i$ iff $s_1^i, s_2^i \in S^i$ and $s_1^i \xrightarrow{act} s_2^i$, where $act \in Act$, (iv) Π^i is the set of atomic propositions defined for robot i, (v) $L^i : S^i \rightarrow 2^{\Pi^i}$ is a map which provides the set of atomic propositions satisfied at a state, (vi) $w^i : E^i \rightarrow \mathbb{N}_{>0}$ is a weight function.

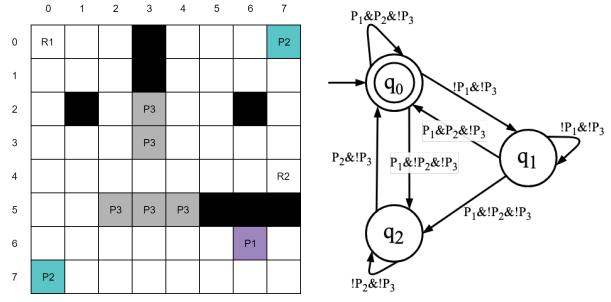
2.3 Joint Transition System

A joint transition system T is a transition system which captures the collective motion of a team of n robots in a workspace($\mathcal{W}$), where each robot executes one action from the set of actions Act available to it. We define a joint transition system as

$$T := (S_T,\ s_0,\ E_T,\ \Pi_T,\ L_T,\ w_T),$$

where (i) S_T is the set of vertices/states in a joint transition system, where each vertex is of form $\langle s^1, s^2, ..., s^n \rangle$, s^i represents the state of robot i in transition system T^i, (ii) $s_0 := \langle s_0^1, s_0^2, ..., s_0^n \rangle \in S_T$ is the joint initial state of the team of n robots, (iii) $E_T \subseteq S_T \times S_T$ is the set of edges. $(s_1, s_2) \in E_T$ iff $s_1, s_2 \in S_T$ and $(s_1^i, s_2^i) \in E^i$ for all $i \in \{1, 2, ..., n\}$, (iv) $\Pi^T := \bigcup_{i=1}^{n} \Pi^i$ is the set of atomic propositions, (v) $L_T : S_T \rightarrow 2^{\Pi_T}$, and $L_T(s_1) := \bigcup_{i=1}^{n} L^i(s_1^i)$ gives us set of propositions true at state s_1, (vi) $w_T : E^T \rightarrow \mathbb{N}_{>0}$, and $w_T(s_1, s_2) := \sum_{i=1}^{n} w^i(s_1^i, s_2^i)$ is a weight function.

We can also think of the transition system as a weighted directed graph with vertices, edges and a weight function. So, whenever we use some graph algorithm over a transition system, we mean to apply it over it's equivalent graph.



(a) Workspace $\mathcal{W}$ with propositions P_1, P_2 and P_3

(b) Büchi automaton B for query: $\square(\diamond P_1 \wedge \diamond P_2 \wedge \neg P_3)$

Figure 1: Workspace $\mathcal{W}$ and Büchi Automaton

EXAMPLE **1** *Throughout this paper, we will use the workspace $\mathcal{W}$ shown in Figure 1(a) for the illustration purpose. We build a transition systems T^i for all the robots over $\mathcal{W}$ where $\Pi^i = \Pi_T = \{P_1, P_2, P_3\}$. The proposition P_i is satisfied if the robot is at one of the locations denoted by P_i. We assume that from any cell in $\mathcal{W}$ a robot can move to one of its neighbouring four cells with cost 1 or stay at the same location with cost 0. Cells with black colour represent obstacles ($\mathcal{O}$).*

2.4 Linear Temporal Logic

The motion planning query/task in our work is given in terms of formulas written using *Linear Temporal Logic* (LTL). LTL formulae over the set of atomic propositions Π_T are formed according to the following grammar (Baier and Katoen 2008):

$$\Phi ::= \texttt{true} \mid a \mid \phi_1 \wedge \phi_2 \mid \neg\phi \mid X\phi \mid \phi_1 U \phi_2.$$

The basic ingredients of an LTL formulae are the atomic propositions $a \in \Pi_T$, the Boolean connectors like conjunction $\wedge$ and negation $\neg$, and two temporal operators X (next) and U (until). The semantics of an LTL formula is defined over an infinite trajectory σ. The trajectory σ satisfies a formula ξ, if the first state of σ satisfies ξ. The logical operators conjunction $\wedge$ and negation $\neg$ have their usual meaning. For an LTL formula ϕ, $X\phi$ is true in a state if ϕ is satisfied at the next step. The formula $\phi_1\ U\ \phi_2$ denotes that ϕ_1 must remain true until ϕ_2 becomes true at some point in future. The other LTL operators that can be derived are $\square$ (*Always*) and $\diamond$ (*Eventually*). The formula $\square\phi$ denotes that the formula ϕ must be satisfied all the time in the future. The formula $\diamond\phi$ denotes that the formula ϕ has to hold sometime in the future. We have denoted negation $\neg P$ as $!P$ and conjunction as $\&$ in the Figures.

2.5 Büchi Automaton

For any LTL formulae Φ over a set of propositions Π_T, we can construct a Büchi automaton with input alphabet $\Pi_B = 2^{\Pi_T}$. We can define a Büchi automaton as

$$B := (Q_B, q_0, \Pi_B, \delta_B, Q_f),$$

where (i) Q_B is a finite set of states, (ii) $q_0 \in Q_B$ is the initial state, (iii) $\Pi_B = 2^{\Pi_T}$ is the set of input symbols accepted by the automaton, (iv) $\delta_B \subseteq Q_B \times \Pi_B \times Q_B$ is a transition relation, and (v) $Q_f \subseteq Q_B$ is a set of final states. An accepting state in the Büchi automaton is the one that needs to occur infinitely often on an infinite length string consisting of symbols from Π_B to get accepted by the automaton.

EXAMPLE **2** *Figure 1(b) shows the Büchi automaton for an LTL task $\square(\diamond P_1 \wedge \diamond P_2 \wedge \neg P_3)$. Here, q_0 is the start state as well as the final state. It informally depicts the steps to be followed in order to compete the task Φ. The transitions $q_1 \rightarrow q_2 \rightarrow q_0$ leads us to visit a state where $P_1 \wedge \neg P_2 \wedge \neg P_3$ is satisfied by going through only those states which satisfy $\neg P_1 \wedge \neg P_3$ and then go to state where $P_2 \wedge \neg P_3$ is satisfied using states which satisfy $\neg P_2 \wedge \neg P_3$. This way, we can also understand the meaning of the other transitions.*

2.6 Product Automaton

The product automaton P between the joint transition system T and the Büchi automaton B is defined as

$$P := (S_P, S_{P,0}, E_P, F_P, w_p),$$

where (i) $S_P = S_T \times Q_B$, (ii) $S_{P,0} := (s_0, q_0)$ is an initial state, (iii) $E_P \subseteq S_P \times S_P$, where $((s_i, q_k), (s_j, q_l)) \in E_P$ if and only if $(s_i, s_j) \in E_T$ and $(q_k, L_T(s_j), q_l) \in \delta_B$, (iv) $F_P := S_T \times Q_f$ set of final states, and (v) $w_P : E_P \rightarrow \mathbb{N}_{>0}$ such that $w_P((s_i, q_k), (s_j, q_l)) := w_T(s_i, s_j)$. To generate

a trajectory in T which satisfies LTL query, we can refer P. Refer (Smith et al. 2010) for examples on product automaton/graph.

3 Problem Definition

Consider a team of robots represented as transition systems $\{T_1, ..., T_n\}$, moving in a static workspace $\mathcal{W}$ and their collective motion is modeled as a joint transition system T. A run over the transition system T starting at initial state s_0 defines the trajectory of the robots in the $\mathcal{W}$. Suppose, the robots are given a task in the form of an LTL query ϕ over Π_T which needs to be completed collectively by them repetitively and infinitely many times. We construct a Büchi automaton B from ϕ. Let $\Pi_c = \{c \mid c \in \Pi_B$ and $\exists \delta_B(q_i, c) = q_j$ where, $q_i \in Q_B$ and $q_j \in Q_f\}$. Let $F_\pi = \{s_i \mid s_i \in S_T$ and $s_i \vDash \pi_j$ where $\pi_j \in \Pi_c\}$. F_π represents a set of all the possible final states (final state is the last state in the complete trajectory in T which satisfies ϕ) to be visited by the team on the path to complete the task. Our objective is to find the path in T(which represents the trajectories of the robots in $\mathcal{W}$) in the form of cycle with minimum cost and also which completes the task. Such path will always contain one of the states from F_π.

Let us assume that there exists at least one run over T which satisfies ϕ. Let $\mathcal{R} = s_0, s_1, s_2, ...$ be an infinite length run/path over T which satisfies ϕ and so there exists $f \in F_\pi$ which occurs on $\mathcal{R}$ infinitely many times. From $\mathcal{R}$, we can extract all the time instances at which f occurs. Let $t^f_{\mathcal{R}}(i)$ denotes the time instance of i^{th} occurrence of state f on $\mathcal{R}$. Our goal is to synthesize an infinite run $\mathcal{R}$ which satisfies the LTL formulae ϕ and minimizes the cost function

$$\mathcal{C}(\mathcal{R}) = \limsup_{i \to +\infty} \sum_{k=t^f_{\mathcal{R}}(i)}^{t^f_{\mathcal{R}}(i+1)-1} w_T(s_k, s_{k+1}) \quad (1)$$

3.1 Prefix-Suffix Structure

The accepting run $\mathcal{R}$ of infinite length can be divided into two components namely *prefix* ($\mathcal{R}_{pre}$) and *suffix* ($\mathcal{R}_{suf}$). A prefix is a finite run from initial state of the robot to an accepting state $f \in F_\pi$ and a suffix is a finite length run starting and ending at f reached by the prefix, and containing no other occurrence of f. This suffix will be repeated periodically and infinitely many times to generate an infinite length run $\mathcal{R}$. So, we can represent run $\mathcal{R}$ as $\mathcal{R}_{pre}.\mathcal{R}^\omega_{suf}$, where ω denotes the suffix being repeated infinitely many times.

Lemma 3.1: For every run $\mathcal{R}$ which satisfies LTL formulae ϕ and minimizes cost function (1), there exists a run $\mathcal{R}_c$ which satisfies ϕ, minimizes cost function (1) and is in prefix-suffix structure. Refer (Smith et al. 2010) for the proof.

The cost of such run $\mathcal{R}_c$ is the cost of its suffix. So, now our goal translates to determining an algorithm which finds minimum cost suffix run starting and ending at a state $f \in F_\pi$ and having a finite length prefix run starting at initial state $s_0 \in S_T$ and ending at f. So, let $\mathcal{R} = \mathcal{R}_{pre}.\mathcal{R}^\omega_{suf}$, where $\mathcal{R}_{pre} = s_0, s_1, s_2, ..., s_p$ be a prefix and $\mathcal{R}_{suf} = s_{p+1}, s_{p+2}, ..., s_{p+r}$ be a suffix, where $s_{p+r} = s_p$. We can redefine the cost function given in 1 as

$$\mathcal{C}(\mathcal{R}) = \mathcal{C}(\mathcal{R}_{suf}) = \sum_{i=p+1}^{p+r-1} w_T(s_i, s_{i+1}) \quad (2)$$

QUESTION **1** *Given a joint transition system T capturing the motion of the team of robots in workspace $\mathcal{W}$ and an LTL formulae ϕ representing the task given to the robots, find an infinite length run $\mathcal{R}$ in prefix-suffix form over T which minimizes the cost function* (2)*.*

3.2 Baseline Solution Approach

The state-of-the-art solution to the above problem (Ulusoy et al. 2013) uses the automata-theoretic model checking approach. It computes product automaton of T and B and then uses Dijkstra's shortest path algorithm to compute the required minimum cost suffix run having a valid prefix. The size of T increases exponentially with the increase in the number of robots and also the workspace size. It consumes a huge amount of memory, which becomes an hindrance to scale up this algorithm. In the next section, we present our algorithm MT* and use the algorithm proposed in (Ulusoy et al. 2013) as the baseline for quantitative comparison.

4 MT* Algorithm

A* algorithm has not been utilized in temporal logic path planning (which includes job sequencing, repetition, synchronization etc.) as there is no notion of a single destination state in LTL motion planning. In MT*, we have found a way to incorporate the heuristic information from the discrete workspaces systematically to achieve substantial speed up in terms of computation time over the baseline solution. Moreover, MT* only computes a reduced version of the product graph which we call the *Abstract Reduced Graph* G_r. Its size is significantly small compared to the product graph P and thus results in faster run times and lower memory consumption.

4.1 Abstract Reduced Graph

We explain the intuition behind the construction of abstract reduced graph G_r using a single robot example.

EXAMPLE **3** *Consider a robot moving in workspace $\mathcal{W}$ shown in Figure 1(a) and has been given an LTL task $\square(\diamond P_1 \wedge \diamond P_2 \wedge \neg P_3)$ whose Büchi automaton B is shown in Figure 1(b). Let T^1 be the transition system of the robot constructed from $\mathcal{W}$. As there is only one robot, the joint transition system T will be same as T^1. Consider a product automaton P of T and B. Suppose $s_0 = \langle(4,7)\rangle$ and therefore $S_{P,0} = (\langle(4,7)\rangle, q_0)$. Now, from here, we must use the transitions in Büchi automaton to find the path in T in the prefix-suffix form. Suppose we find such a path on which we move to state $(\langle(4,6)\rangle, q_1)$ from $(\langle(4,7)\rangle, q_0)$ as per the definition of the product automata. From $(\langle(4,6)\rangle, q_1)$, we must visit a location where $P_1 \wedge \neg P_2 \wedge \neg P_3$ is satisfied so that we can move to Büchi state q_2 from q_1. All the intermediate states till we reach such a state must satisfy $\neg P_1 \wedge \neg P_3$ formulae. Suppose we next move from $(\langle(4,6)\rangle, q_1)$ to $(\langle(0,2)\rangle, q_2)$ on P which satisfies $P_1 \wedge \neg P_2 \wedge \neg P_3$ and this path is $(\langle(4,6)\rangle, q_1) \rightarrow (\langle(4,5)\rangle, q_1) \rightarrow (\langle(4,4)\rangle, q_1) \rightarrow ... \rightarrow (\langle(1,2)\rangle, q_1) \rightarrow (\langle(0,2)\rangle, q_2)$. On the path from $(\langle(4,6)\rangle, q_1)$ to $(\langle(0,2)\rangle, q_2)$, all the intermediate nodes satisfy self loop transition condition on q_1. As an analogy, we can consider the self loop transition condition $\neg P_1 \wedge \neg P_3$ over q_1 as the constraint which must be followed by the intermediate states while completing a task of moving to the location which satisfies the transition condition from q_1 to q_2, i.e. the self loop is the only means to navigate to the next state. Using this as an abstraction method over the product automaton, we directly add an edge from state $(\langle(4,6)\rangle, q_1)$ to state $(\langle(0,2)\rangle, q_2)$ in the reduced graph assuming that there exists a path between these two states and we will explore*

Algorithm 1: MT*

Input: Transition systems $\{T^1, ..., T^n\}$, ϕ: LTL formulae
Output: A run $\langle \mathcal{R}^1, ..., \mathcal{R}^n \rangle$ that satisfies ϕ

$B(Q_B, q_0, \Pi_B, \delta_B, Q_f) \leftarrow$ `ltl_to_Buchi` (ϕ)
for *all* $q_i, q_j \in Q_B$, *where* $\delta_B(q_i, c_{pos}) = q_j$ **do**
  $S_*[c_{pos}]$ = `Abstract_Distant_Neighbours`(c_{pos})
$G_r(S_r, v_0, E_r, F_r, \mathcal{N}) \leftarrow$ `Generate_Redc_Graph`$(\mathtt{B}, \mathtt{T})$
for *all* $f \in F_r$ **do**
  for *each simple cycle* C_f *containing* f *in* G_r **do**
    $B' \leftarrow$ `Extract_Buchi_Trans_From_`$\mathtt{C_f}(C_f, B)$
    for $i \leftarrow \{1, .., n\}$ **do**
      $c_f^i \leftarrow$ `Project_`$\mathtt{C_f}$`_Over_`$\mathtt{T^i}(C_f, i, G_r)$
      $B^i \leftarrow$ `Project_`$\mathtt{C_f^i}$`_Over_`$\mathtt{B'}(c_f^i, B', G_r)$
      $\mathcal{R}_f^i \leftarrow$ `Optimal_Run`(B^i, c_c^i, T^i)
    $\mathcal{R}_f^{suf}(\langle \mathcal{R}_s^1, .., \mathcal{R}_s^n \rangle) \leftarrow$ `Sync`$(\langle \mathcal{R}_f^1, .., \mathcal{R}_f^n \rangle)$
    $\mathcal{R}_f^{pre}(\langle \mathcal{R}_p^1, .., \mathcal{R}_p^n \rangle) \leftarrow$ `Compt_Prefix`(f, B, G_r)
$\mathcal{R}_P^{suf} \leftarrow \underset{\mathcal{R}_f^{suf} \text{ with a valid prefix}}{\operatorname{argmin}} \mathcal{C}\left(\mathcal{R}_f^{suf}\right)$
$\mathcal{R}_P^{pre} \leftarrow$ prefix of $\mathcal{R}_P^{suf}$
$\mathcal{R}_P = \mathcal{R}_P^{pre}.\mathcal{R}_P^{suf}$
Project $\mathcal{R}_P$ over T to compute $\mathcal{R}_T$
Project $\mathcal{R}_T$ over $\{T^1, ..., T^n\}$ to obtain runs $\langle \mathcal{R}^1, ..., \mathcal{R}^n \rangle$

Procedure ***Generate_Redc_Graph***(B, T)
  $v_{init} \leftarrow v_0(s_0, q_0)$
  let Q be a queue data-structure
  Initialize empty reduced graph G_r
  label v_{init} as discovered and add it to S_r
  Q.`enqueue`(v_{init})
  while Q *is not empty* **do**
    $v_i(s_i, q_i) \leftarrow Q$.`dequeue`$()$
    if $\exists \delta_B(q_i, c_{neg}) = q_i$ *and* $\nexists(\delta_B(q_i, c_{neg}) = q_j$ *such that* $q_i \neq q_j)$ **then**
      for *all* $v_l(s_l, q_l)$ *such that* $s_l \in S_*[c_{pos}]$ *and* $\delta_B(q_i, c_{pos}) = q_l$ **do**
        $E_r \leftarrow (v_i, v_l)$
        $\mathcal{N}(v_i, v_l) \leftarrow false$
        if v_l *is not labelled as discovered* **then**
          label v_l as *discovered*, add it to S_r
          Q.`enqueue`(v_l)
    else
      for *all* $v_l(s_l, q_l)$ *such that* $s_l \in S_*[c_{pos}]$ *and* $\delta_B(q_i, c_{pos}) = q_l$ **do**
        $E_r \leftarrow (v_i, v_l)$
        $\mathcal{N}(v_i, v_l) \leftarrow true$
        if v_l *is not labelled as discovered* **then**
          label v_l as *discovered*, add it to S_r
          Q.`enqueue`(v_l)
      for *all* q_l *such that* $\exists \delta_B(q_i, c_{neg}) = q_l$ **do**
        $v_l \leftarrow (s_*, q_l)$
        $E_r \leftarrow (v_i, v_l)$
        $\mathcal{N}(v_i, v_l) \leftarrow true$
        if v_l *is not labelled as discovered* **then**
          label v_l as *discovered*, add it to S_r
          Q.`enqueue`(v_l)
  return G_r

this path opportunistically only when it is required. This is the first main idea behind MT algorithm.*

Throughout this paper, we call an atomic proposition with negation a *negative proposition* and an atomic proposition without negation a *positive proposition*. For example, $\neg P_2$ is a negative proposition and P_2 is a positive proposition. We divide the transition conditions in B into two types. A transition condition which is a conjunction of all negative propositions is called a *negative transition condition* and is denoted by c_{neg}. The one which is not negative is called a *positive transition condition*, and is denoted by c_{pos}. For example, $\neg P_1 \wedge \neg P_3$ is a negative transition condition whereas $P_1 \wedge \neg P_2 \wedge \neg P_3$ is a positive transition condition. For every transition, we can consider all the positive propositions as a task to be completed by all the robots collectively and all the negative transitions as constraints that must be followed by all the robots. For example, in $P_1 \wedge P_2 \wedge \neg P_3$, some robot must visit a location where P_1 is satisfied. At the same time, some robot must visit a location where P_2 is satisfied and all the robots must satisfy $\neg P_3$ to satisfy transition $P_1 \wedge P_2 \wedge \neg P_3$ completely and collectively. Now we explain the second main idea behind MT* .

EXAMPLE **4** *Consider that $c_{pos} = P_2 \wedge \neg P_3$ be a transition condition from B. Suppose here we are planning path for two robots. So, to satisfy this transition, one of the robots must go to a location where $P_2 \wedge \neg P_3$ is satisfied. There are two ways to achieve it. First one is when robot 1 reaches a location on $\mathcal{W}$ where $P_2 \wedge \neg P_3$ is satisfied and robot 2 is at location where $\neg P_3$ is satisfied. Then we can say that $P_2 \wedge \neg P_3$ satisfied by both the robots. And second one is when robot 2 satisfies $P_2 \wedge \neg P_3$ and robot 1 satisfies $\neg P_3$. In the first case, robot 1 could be at either $(0, 7)$ or $(7, 0)$ which satisfy $P_2 \wedge \neg P_3$ and robot two could be at any of 52 locations which are obstacle free and satisfy $\neg P_3$. The number of ways in which two robots can satisfy the first case are $52 * 2 = 104$. Similarly, in another 104 ways the robots can satisfy second case. We represent all these $104 + 104 = 208$ satisfying configurations symbolically as $\{\langle(0, 7), (*, *)\rangle, \langle(7, 0), (*, *)\rangle, \langle(*, *), (0, 7)\rangle, \langle(*, *), (7, 0)\rangle\}$. Here, $((0, 7), (*, *))$ says that robot 1 is at location $(7, 0)$ whereas robot 2 could be at any obstacle-free location which satisfies $\neg P_3$. We call this set as abstract neighbour set for transition condition c_{pos} and represent it as $S_*[c_{pos}]$. For a single robot, we represent an unknown state symbolically as $s_*^i := (*, *)$, where $i \in \{1, ..., n\}$. If the locations of all the robots are unknown, then we represent such state as $s_* := \langle s_*^1, ..., s_*^n \rangle$. We can easily compute $S_*[c_{pos}]$ for any c_{pos} transition condition by distributing all the task propositions among the robots in all possible ways and making location for robots who doesn't receive any task from c_{pos} as $(*, *)$. While computing S_*, We also store the tasks and the constraints that each robot satisfy in a particular configuration using a map L_*. For example, $S_*[c_{pos} = P_2 \wedge \neg P_3] = \{\langle(0, 7), (*, *)\rangle, \langle(7, 0), (*, *)\rangle, \langle(*, *), (0, 7)\rangle, \langle(*, *), (7, 0)\rangle\}$ and $L_*[((0, 7), (*, *))] = \{\{P_2, \neg P_3\}, \{\neg P_3\}\}$ and so on.*

Using the above ideas, we can represent the product graph symbolically as a significantly small abstract reduced graph. While constructing an abstract reduced graph, we add an edge from node $v_i(s_i, q_i)$ to some node using following rules:

$Condition$: $\exists \delta_B(q_i, c_{neg}) = q_i$ **and** $\nexists \delta_B(q_i, c_{neg}) = q_j, q_i \neq q_j$, *i.e.*, if there exists a negative self loop over q_i and there does not exist any negative transition from q_i to some other state in the Büchi automaton.

1. ***If Condition is true then*** add an edge from $v_i(s_i, q_i)$ to all

$v_l(s_l, q_l)$ such that $\exists \delta_B(q_i, c_{pos}) = q_l$ and $s_l \in S_*[c_{pos}]$. Here, q_i and q_l can be the same. In short, in this condition we add all the nodes as neighbours to v_i which satisfy an outgoing c_{pos} transition from q_i and skip nodes which satisfy c_{neg} self loop transition assuming that c_{neg} self loop transition can be used to find the actual path from v_i to v_l later in the algorithm. We use $\mathcal{N}$ to keep track neighbour information in G_r. $\mathcal{N}(v_a, v_b) = true$ says that v_b must be a neighbour of v_a in T. In this condition, we set $\mathcal{N}(v_i, v_l)$ to $false$. It says that v_i and v_l may not be actual neighbours in the actual product graph P. For example, consider an edge between $v_i = (\langle(6,6)(*,*)\rangle, q_2)$ and $v_l = (\langle(7,0)(*,*)\rangle, q_0)$ in Figure 2. Here, we say that the path between $(6,6)$ and $(7,0)$ could be established through c_{neg} transition condition. Similarly, whatever value we fill for unknowns $(*,*)$, intermediate path between the two must satisfy c_{neg} condition. We call this way of adding neighbours as *distant neighbour way/condition*.

2. ***If Condition is false then*** the same as above, we add an edge from v_i to all $v_l(s_l, q_l)$ such that $\exists \delta_B(q_i, c_{pos}) = q_l$ and $s_l \in S_*[c_{pos}]$. And for all c_{neg} outgoing transitions from q_i to some q_l, we add an edge from v_i to $v_m(s_*, q_l)$. Here $s_* := \langle s_*^1, ..., s_*^n \rangle$ where, $s_*^i = (*,*)$. Here, q_i and q_l can be the same. We call this way of adding neighbours as *product automata way/condition*. For all these transitions, we set $\mathcal{N}(v_i, v_l)$ to $true$ which says that v_i and v_l are actual neighbours in T. At this moment we do not know the complete value of the nodes in v_i and v_l. For example, as shown in Figure 2, v_i could be$(\langle(*,*)(7,0)\rangle, q_0)$ and v_l could be $(\langle(6,6)(7,0)\rangle, q_0)$. Here for robot 2, $(7,0)$ from v_i is the neighbour of $(7,0)$ form v_l. But, robot 1's location in v_i is not known (represented as $(*,*)$) and is known in v_l as $(6,6)$. So, later in the algorithm, whenever we fill this unknown value, we have to ensure that it is neighbour of $(6,6)$ in T[1].

Now we formally define the *Abstract Reduced Graph* for the transition system T and the Büchi automaton B as

$$G_r := (S_r, v_{init}, E_r, F_r, \mathcal{N})$$

where (i) S_r the set of vertices added as per the above rules, (ii) $v_{init} = (s_0, q_0)$ is an initial state, (iii) $E_r \subseteq S_r \times S_r$, is a set of edges added as per the above conditions, (iv) the set of final states $F_r \subseteq S_r$ and $v_i(s_i, q_i) \in F_r$ iff $q_i \in Q_B$, and (v) $\mathcal{N}$ stores the neighbour information.

In procedure `Generate_Redc_Graph` of Algorithm 1, we run the Breadth-First-Search (BFS) algorithm starting from node (s_0, q_0) and add the neighbours using the rules mentioned above.

EXAMPLE **5** *The abstract reduced graph generated for a two robot system over a workspace $\mathcal{W}$ and Büchi automaton B from Figure 1 is shown in Figure 2. All the nodes enclosed in a rectangle represent an abstract neighbour set S_* for some transition. The edges show the transition conditions written on them. N mentioned on the transition represents $\mathcal{N}$ value for that transition. * on the transition says that this transition is applicable only if the nodes are actual neighbours (as we wanted to show one edge for all the nodes in a rectangle).*

Consider an edge from $v_1 = (\langle(0,0),(4,7)\rangle, q_0)$ to $v_2 = (\langle(6,6),(0,7)\rangle, q_0)$ with transition condition $P_1 \wedge P_2 \wedge \neg P_3\{N\}^*$. Here $\{N\}$ says that v_1 and v_2 should be neighbours but as these two are not neighbours of each other in T, an edge cannot exist between v_1 and v_2. The symbol * here says that the edge from $(\langle(0,0),(4,7)\rangle, q_0)$ to the other nodes is valid if the other nodes are neighbours of $(\langle(0,0),(4,7)\rangle, q_0)$. In Figure 2, the transition condition accompanied with c_{neg} condition in curly braces $\{\}$ represents a c_{neg} transition that must be used to reach next node. In every node, below the robot co-ordinates, we show the L_* value for that node.

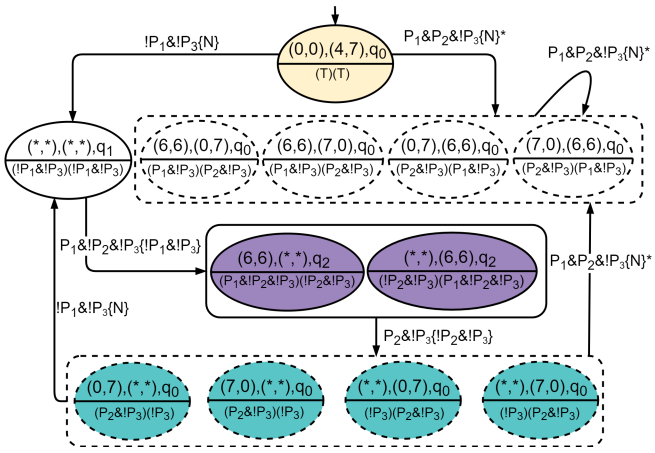


Figure 2: Abstract Reduced Graph for a 2 Robot System Having Workspace and Büchi Automaton from Figure 1

We explain the construction of the Abstract Reduced Graph shown in Figure 2. Initially, robot 1 is at location $(0,0)$ and robot 2 is at $(4,7)$. So, here $s_0 = \langle s_0^1, s_0^2 \rangle = \langle(0,0),(4,7)\rangle$. We start BFS with node $v_0 = (s_0, q_0) = (\langle(0,0),(4,7)\rangle, q_0)$. For the first time $v_i = v_0$ is de-queued from the queue Q, so we add the neighbours of v_0 to G_r. Here, q_0 does not have c_{neg} type self transition loop in B. So, we first add all c_{pos} satisfying nodes as neighbours of v_0. Here, there exists a transition form q_0 to q_0 on $c_{pos} = P_1 \wedge P_2 \wedge \neg P_3$. We add edges from v_0 to all the nodes in $S_*[(P_1 \wedge P_2 \wedge \neg P_3)]$ which are $\langle(6,6)(0,7)\rangle, \langle(6,6)(7,0)\rangle, \langle(0,7)(6,6)\rangle$, and $\langle(7,0)(6,6)\rangle$ with q_0 as the Büchi state. As these nodes have been added as per product automaton condition, these nodes must be neighbours to v_0. However, None of these nodes are neighbours to v_0. So none of these edges will be actually added to G_r. We have only shown these transitions in Figure 2 for the sake of completeness and understanding of the readers. This kind of uncertainty we represent using $*$ on the transition condition. Now, there also exists a transition from q_0 to q_1 with transition condition $\neg P_1 \wedge \neg P_2$. And as this is a c_{neg} type transition, we add an edge from v_0 to $(\langle(*,*),(*,*)\rangle, q_1)$. Again this node has been added as per product automaton condition, so whatever value we choose to put in place of $\langle(*,*)(*,*)\rangle$ must be neighbour of v_0. Now, suppose $v_i = (\langle(*,*),(*,*)\rangle, q_1)$ has been de-queued from the queue Q. Here, there exists a negative type self loop over q_1 with transition condition $c_{neg} = \neg P_1 \wedge \neg P_3$ and there does not exist any c_{neg} transition from q_1 to any other state in B. So, we add all the nodes as neighbours to v_i which satisfy c_{pos} transition conditions and ignore c_{neg} self loop. There exists a positive transition from q_1 to q_2 with transition condition $c_{pos} = P_1 \wedge \neg P_2 \wedge \neg P_3$. So, we add edges from v_i to all the nodes in $S_*[(P_1 \wedge \neg P_2 \wedge \neg P_3)]$ which are $\langle(6,6),(*,*)\rangle, \langle(*,*),(6,6)\rangle$ with Büchi state q_2. Here, as these nodes have been added as neighbours to v_i using

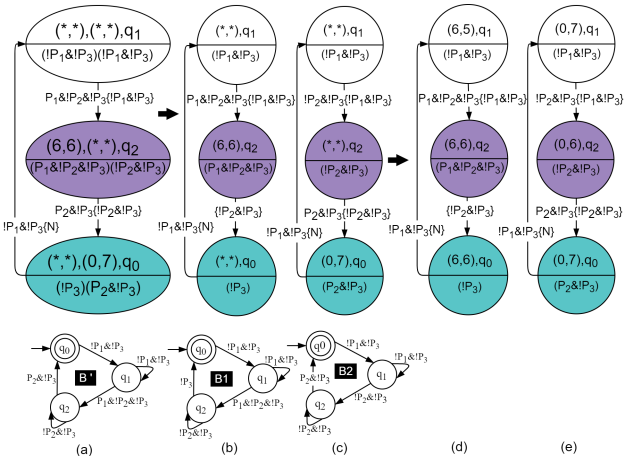


Figure 3: Suffix Computation in MT* Algorithm

distant neighbour condition, these nodes may not be actual neighbours of v_i in P. We explain this in more detail in the next section. Like this we add nodes to G_r. The completed G_4 is shown in Figure 2.

4.2 MT* Procedure

We outline all the block steps of MT* in Algorithm 1. We explain only the concepts for the major steps and not the complete algorithm in this section due to space constraints. We provide the link to the implementation for reference in the evaluation section.

The inputs to the algorithm are the transition systems for n robots $\{T^1, ..., T^n\}$ and an LTL query ϕ. The goal of the algorithm is to compute a minimum cost run $\mathcal{R}^i$ over T^i for n robots such that if they follow these trajectories, they should be able to complete the task Φ collectively. These trajectories are in the form of a prefix and a suffix. The robots use the prefix to reach suffix from an initial location s_0. Then they execute the suffix run over $\mathcal{W}$ repetitively and infinitely many times to complete the given task repetitively and infinitely often. (1) In this algorithm, we first compute the Büchi automaton from the given LTL task ϕ. (2) Then for all c_{pos} transition conditions present in B, we compute the abstract neighbour set $S_*[c_{pos}]$ using the procedure `Abstract_Distant_Neighbour()` as explained in section (4.1). (3) Then we generate Abstract Reduced Graph(G_r) using procedure `Generate_Redc_Graph()` from Algorithm 1. (4) Final state $f_p \in F_P$ corresponds to an incoming transition to a final state in the Büchi automaton. In the product graph, the final state corresponds to a state where the task completes and starts again. As our task is to find a minimum cost cyclic trajectory which satisfies LTL task ϕ, it will always contain a final state and as the trajectory is cyclic, if we start from f_p, we will again come back to it over the cyclic trajectory. F_r is a symbolic representation of F_P. So, for each state $f \in F_r$, we compute all the possible cycles starting and ending at f. We choose the one with the minimum cost and reachable from v_{init} as our final solution. (5) *Optimization*: If a node other than f repeats on the suffix cycle, then there exists another cycle on suffix. We can always obtain a suffix with smaller cost by removing this extra cycle. So, we only search for simple cycle (in which no node is repeated except the starting and the ending node f). We can easily find such cycles using DFS algorithm starting with node f. (6) For each $f \in F_r$ and for each simple cycle C_f starting and ending at f in G_r, we follow the following steps. We will also use example mentioned in Figure 3 for better understanding of the readers. (i) `Extract_Buchi_Trans_From_c_f( )`: Each cycle C_f in G_r also represents a cycle of transitions in the Büchi automaton. For example, consider a cycle C_f starting at ending at $f = (\langle(*,*),(0,7)\rangle, q_0)$ in Figure 3(a). From this C_f, we can extract an automaton B' which is a subgraph of B and also shown in Figure 3(a). (ii) After this, we decouple the joint suffix cycle into n cycles individual to each robot. (iii) `Project_C_f_Over_T^i( )`: We use this procedure to project/extract C_f over T^i to compute the trajectory c_f^i for robot i. In Figure 3, We decouple joint trajectory shown in sub-figure (a) into two trajectories c_f^1 and c_f^2 shown in sub-figure (b) and (c) respectively. Transitions are also divided as per the constraints allocated to the individual robots. (iv) `Project_C_f^i_Over_B'( )`: Using this procedure, we derive n automatons $\{B^1, ..., B^n\}$ from the transitions of cycles $\{c_f^1, ..., c_f^n\}$. These automatons represent the path/constraints that each robot must follow in this particular joint trajectory/task C_f. Automatons B^1 and B^2 are shown in Figure 3 (b) and (c). (v) `Optimal_Run( )`: In this procedure, we complete the individual incomplete trajectories c_f^i for all the robots using their individual automatons B^i. For example, the incomplete trajectory shown in sub-figure (b) is completed using automata B^1 to obtain the completed trajectory shown in sub-figure (d). In trajectory (b), we first find the first known node which is $s^1_{source} = (\langle(6,6)\rangle, q_0)$. Then we find the next known node on the cyclic trajectory which is also $s^1_{dest} = (\langle(6,6)\rangle, q_0)$. Then we attempt to find the path from s^1_{source} to s^1_{dest} with automata B^1 in T^i using single robot LTL path finding algorithms T* (Khalidi and Saha 2018) or *Optimal_Run* (Smith et al. 2010). In these algorithms, as we now have concrete goal node, we can use A* instead of Dijkstra's Algorithm to improve the computation time. Here, there was only one known node in the trajectory. However, in general, we continue like this till we find all the unknown sub-trajectories in the suffix cycle. (vi) This way, the problem of finding the joint trajectory for n robots has been reduced to finding n trajectories for a single robot systems. As, the size of the single robot transition system is much smaller than the joint transition system, MT* produces results much faster than the state of art algorithm (Ulusoy et al. 2013). (vii) `Sync( )`: When we generate the robot trajectories independently for individual robots, the generated trajectories may not be in sync i.e. the computed individual robot trajectories can be of different lengths and because of this, the sequence in which particular locations are visited may change and may not satisfy ϕ. However, if the robots can be stopped at some location during their operation, it is straight-forward to achieve synchronization. There are two types of transitions in G_r. One is added using the product automata condition, in which both the nodes should be neighbours. In this case, all the generated individual trajectories will have consistent Büchi state. Second one is the transition added using the distant neighbour condition, in which we assume that the added node will be reached using c_{neg} type self loop. In such cases, generated individual trajectories could be of different length.

EXAMPLE **6** *Consider an example of some hypothetical LTL query in which there is an edge from* $(\langle(6,6)(6,6)\rangle, q_1) \xrightarrow{P_2 \wedge \neg P_3} (\langle(7,0),(0,7)\rangle, q_2)$ *in* G_r *with* $c_{neg} = \neg P_2 \wedge \neg P_3$ *type negative self loop over* q_1. *In this scenario, individual trajectories generated*

using procedure `Optimal_Run()` *from* $(\langle(6,6)\rangle, q_1)$ *to* $(\langle(7,0)\rangle, q_2)$ *for robot 1 is* $\rho_1 = (\langle(6,6)\rangle, q_1) \xrightarrow{\neg P_2 \wedge \neg P_3} (\langle(7,6)\rangle, q_1) \xrightarrow{\neg P_2 \wedge \neg P_3}$ *....(4 nodes)...* $\xrightarrow{\neg P_2 \wedge \neg P_3} (\langle(7,1)\rangle, q_1) \xrightarrow{P_2 \wedge \neg P_3} (\langle(7,0)\rangle, q_2)$ *consisting of total 6 transitions with* c_{neg} *type transition condition and one* c_{pos} *transition* $(P_2 \wedge \neg P_3)$*. Whereas, for robot 2, individual trajectory is* $\rho_2 = (\langle(6,6)\rangle, q_1) \xrightarrow{\neg P_2 \wedge \neg P_3} (\langle(6,5)\rangle, q_1) \xrightarrow{\neg P_2 \wedge \neg P_3}$ *...(14 nodes)...* $\xrightarrow{\neg P_2 \wedge \neg P_3} (\langle(1,7)\rangle, q_1) \xrightarrow{P_2 \wedge \neg P_3} (\langle(0,7)\rangle, q_2)$ *with 16* c_{neg} *transitions and 1* c_{pos} *transition* $(P_2 \wedge \neg P_3)$*. While synchronizing these two trajectories, robot 1 waits at* $(\langle(6,6)\rangle, q_1)$ *in* ρ_1 *(shorter trajectory) to extend it as* $\rho_1' = (\langle(6,6)\rangle, q_1) \xrightarrow{\neg P_2 \wedge \neg P_3} (\langle(6,6)\rangle, q_1) \xrightarrow{\neg P_2 \wedge \neg P_3}$ *...(9 times)...* $\xrightarrow{\neg P_2 \wedge \neg P_3} (\langle(6,6)\rangle, q_1) \xrightarrow{\neg P_2 \wedge \neg P_3} (\langle(7,6)\rangle, q_1) \xrightarrow{\neg P_2 \wedge \neg P_3}$ *...(4 nodes)...* $\xrightarrow{\neg P_2 \wedge \neg P_3} (\langle(7,1)\rangle, q_1) \xrightarrow{P_2 \wedge \neg P_3} (\langle(7,0)\rangle, q_2)$ *to make both the individual trajectories synchronized so that both of them have the same number of transitions for the same transition condition.*

There could be a case, in which all the nodes in the abstract individual trajectory for a particular robot have all nodes as $(*,*)$. In short, this robot is not doing anything in this team task sequence C_f. In such a case, we can directly ignore this robot as it is not doing anything. For example, consider a $C_f = (\langle(7,0,)(*,*)\rangle, q_0) \rightarrow (\langle(*,*)(*,*)\rangle, q_1) \rightarrow (\langle(6,6)(*,*)\rangle, q_2)$ in which all the coordinates for robot 2 are $(*,*)$.

EXAMPLE 7 *Single robot trajectories shown in Figure 3 (d) and (e) have the same number of nodes and also have the same Büchi states. Thus, they are in sync. We can combine them as* $\mathcal{R}_c^{suf} = (\langle(6,5),(0,7)\rangle, q_1) \rightarrow (\langle(6,6),(0,6)\rangle, q_2) \rightarrow (\langle(6,6),(0,7)\rangle, q_0) \rightarrow (\langle(6,5),(0,7)\rangle, q_1)$ *with cost equal summation of individual cost which is* $2+2=4$.

(viii) `Compt_Prefix( )`: After synchronizing the individual trajectories, we now know the exact co-ordinates of the final state $f \in F_r$ in C_f. For example, C_f in 3(a) has $f = (\langle(*,*)(0,7)\rangle, q_0)$ which we computed as $(\langle(6,6)(0,7)\rangle, q_0)$. Now, in this step we can find a prefix path from initial state $v_{init} = (s_0, q_0)$ to this computed f using the same steps we used to compute the suffix (we find path instead of cycle). If the suffix has a valid prefix (i.e. the suffix is reachable from the initial state of the robots), we can consider such suffix a valid suffix.

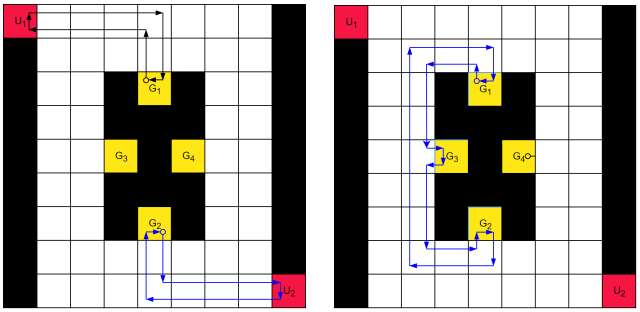
(7) From all those suffix cycles, we choose one with minimum cost and have valid prefix as our final outcome. Project it over $\{T^1, \ldots, T^n\}$ to obtain $\{\mathcal{R}^1, ..., \mathcal{R}^n\}$. For the Abstract Reduced Graph G_r shown in Figure 2, the suffix cycle with the minimum cost is $C_f = (\langle(6,6),(0,7)\rangle, q_0) \rightarrow (\langle(6,6),(0,7)\rangle, q_0)$ with cost 0. The Prefix can be computed accordingly.

Memoization. We can store once generated paths for individual robots and use them later in `Optimal_Run` to reduce the computation time. For example, we can store the actual path computed for the abstract path $(\langle(6,6)\rangle, q_2) \xrightarrow{\neg P_2 \wedge \neg P_3} (\langle(*,*)\rangle, q_0) \xrightarrow{\neg P_1 \wedge \neg P_3\{N\}} (\langle(*,*)\rangle, q_1) \xrightarrow{P_1 \wedge \neg P_2 \wedge \neg P_3\{\neg P_1 \wedge \neg P_3\}} (\langle(6,6)\rangle, q_2)$. This memoization saves lot of extra computation.

Correctness and Optimality. Due to the space constraint, we omit the proof of optimality and correctness of MT* algorithm from this paper. The proofs are available in the full version that is submitted as a supplementary material. There we prove that the trajectory generated by MT* satisfies the given LTL formula and minimizes the cost function given by Equation (2).

5 Evaluation

In this section, we present several results to establish the computational efficiency of MT* algorithm against the baseline solution (Ulusoy et al. 2013). The results have been obtained on a desktop computer with a 3.4 GHz quadcore processor with 16 GB of RAM. We use LTL2TGBA tool (Duret-Lutz and Poitrenaud 2004) as the LTL query to Büchi automaton converter. The C++ implementations of MT* and the Baseline algorithm are available in the following repository for reference: https://www.dropbox.com/sh/vqib2v18w98h4ml/AADIemDSVKW4pviXDWdkRWLca?dl=0.



(a) Trajectory for Query ϕ_2 (b) Trajectory for Query ϕ_5

Figure 4: Generated Trajectories for a Two Robot System

The workspace used in our experiment is a 2-D grid shown in Figure 4 (borrowed from (Ulusoy et al. 2013)). Each grid-cell has 4 neighbours. The cost of each edge between the neighbouring cells is 1 unit. In the workspace, U_1 and U_2 are data upload locations 1 and 2, whereas G_1, G_2, G_3 and G_4 are data gather locations 1 to 4.

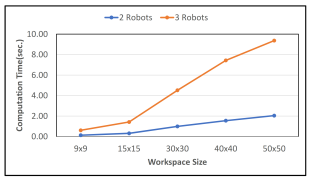
We have evaluated MT* algorithm for five LTL queries borrowed from (Ulusoy et al. 2013). The LTL queries are denoted as $\phi_1, \phi_2, \ldots, \phi_5$. We define propositions over the workspace in a following way. $gather$: Data has been gathered from a gather station, $rXgather$: Robot X has gathered data from a gather station, $gatherY$: Data has been gathered from the gather station Y, $rXgatherY$: Robot X has gathered data from the gather station Y. We define propositions for 'upload' in same way. Due to space constraints, we discuss two of the five queries. Detailed results are available in the full version of the paper submitted as a supplementary material.

Query ϕ_2: Mission is "Each Robot must repeatedly visit a data gather location at same time synchronously to gather data and then upload that data to an upload station before gathering the new data again'

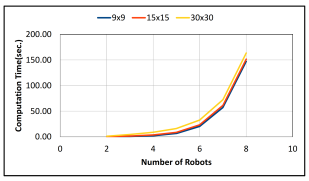
$$\phi_2 = \square\diamond gather \wedge$$
$$\square(r1gather \implies \mathrm{X}(\neg(r1gather)\mathrm{U}(r1upload))) \wedge$$
$$\square(r2gather \implies \mathrm{X}(\neg(r2gather)\mathrm{U}(r2upload))) \wedge$$
$$\square(gather \implies (r1gather \wedge r2gather))$$

Φ	Büchi States	9 × 9 Workspace							15 × 15 Workspace				30 × 30 Workspace			
		2 Robots			3 Robots			8 Robots	2 Robots			8 Robots	2 Robots			8 Robots
		B.S. (sec)	MT* (sec)	Speed Up	B.S. (sec)	MT* (sec)	Speed Up	MT* (sec)	B.S. (sec)	MT* (sec)	Speed Up	MT* (sec)	B.S. (sec)	MT* (sec)	Speed Up	MT* (sec)
ϕ_1	12	20.6	13.6	**1.5**	-	266	**-**	5397.19	3635	19.8	**184**	5513.59	-	37.41	**-**	6022.69
ϕ_2	5	2.8	0.1	**21.9**	9786	0.61	**16037**	147.11	74	0.3	**243**	151.54	985	1.0	**985**	163.47
ϕ_3	5	22.3	0.9	**23.7**	11665	5.5	**2107**	1211.12	496	2.5	**198**	1196.55	9043	8.3	**1086**	1311.95
ϕ_4	5	1.8	0.04	**40.0**	1150	0.1	**11500**	66.36	38	0.06	**664**	66.48	728	0.12	**6129**	67.53
ϕ_5	5	11.4	0.05	**218.8**	-	0.19	**-**	382.33	579	0.07	**7846**	383.66	10937	0.14	**80633**	383.99

Table 1: Baseline Solution(B.S.) Vs MT*



(a) Computation Time(s) Vs Workspace Size



(b) Computation Time(s) Vs No. of Robots

Figure 5: Scalability results for LTL Query ϕ_2

$\|\mathcal{W}\|$	$\|n\|$	P		G_r	
		$\|S_P\|$	$\|E_P\|$	$\|S_r\|$	$\|E_r\|$
9x9	2	9605	305767	15	26
15x15	2	152101	6329687	15	26
30x30	2	2762245	126946183	15	26
40x40	2	9375845	442918092	15	26
9x9	3	480254	60541878	19	66

Table 2: Comparison of No. of Vertices and Edges in Product Graph (P) and Abstract Reduced Graph (G_r) for Query ϕ_2

Query ϕ_5: Mission is "Gather data from all the gather stations"

$\phi_5 = \Box\Diamond gather1 \wedge \Box\Diamond gather2 \wedge \Box\Diamond gather3 \wedge \Box\Diamond gather4$

The generated trajectories can be seen in Figure 4 in which the hollow circle represents the start of the trajectory. For ϕ_2, Robot 1 gathers from G1 and Robot 2 from G2 at the same time to induce the synchronization constraint $\Box(gather \implies (r1gather \wedge r2gather))$. They upload it to the nearest upload station to minimize the total cost and move again to gather data at G1 and G2 respectively to complete the cyclic trajectory. Total cost of the trajectory for both the robots is 24. In ϕ_5, in order to minimize the cost, robot 1 gathers data from G1, then from G3, and then from G2. Whereas robot 2 stays at gather station G4 (Assuming that staying at the same location incurs zero cost). For ϕ_5, total cost of the trajectory is 26 for both the robots.

In Table 1, we list down the computation times of the baseline solution and MT* for queries $\phi_1, ..., \phi_5$. Büchi states column lists the number of states in the Büchi automaton for the corresponding LTL mission. We also list the speed-up that MT* achieves over the baseline solution for the corresponding query. We list these results for different sizes of workspaces. A 9×9 workspace is shown in Figure 4. The 15×15 and 30×30 workspaces are similar to the 9×9 map with the same number data gather and data upload locations. In the table, we can observe significant speed up that MT* achieves over the baseline solution. In the tables, we have shown '-' for the entries which we could not compute due to insufficient RAM (16 GB) or very high computation time (> 10000 sec) . For 8 robots, we have only shown computation time for MT* as we could not generate results for baseline solution beyond 3 robots due to very high computation time and memory requirement.

For the graphs in Figure 5, we have used workspaces 9×9, 15×15, 30×30, 40×40 and 50×50. In Figure 5(a), we observe the performance of MT* with increasing workspace size for a 2 robots and a 3 robots system. We have shown this result for query ϕ_2. The computation time of MT* increases almost linearly for all the LTL queries. This is because the size of the abstract reduced graph remains the same with the increase in the workspace size. The computation time for single robot trajectories in the procedure `Optimal_Run` increases with the increase in the workspace size and thus MT* achieves almost linear increase in the computation time with the increase in the workspace size. These results are consistent for other queries and workspaces.

In Figure 5(b), we observe the performance of MT* with the increase in the number of robots. Computation time of MT* increases exponentially with increase in number of robots for all the LTL queries. Graph shown is for query ϕ_2. This is because, with increase in the number of robots, the number of possible task assignments also increases exponentially.

In Table 2, we show the number of vertices in product graph $|S_P|$, number of edges in Product graph $|E_P|$, number of vertices in Abstract Reduced Graph $|S_r|$, and number of edges in Abstract Reduced Graph $|E_r|$ for different workspace sizes $|\mathcal{W}|$ and different number of robots $|n|$. The Abstract Reduced graph remains the same with the increase in the workspace size and is significantly smaller than the product graph. This is the main reason behind the superior performance of MT* over the baseline solution in terms of computation time. The sizes of the graphs are also a direct indicator of the memory requirement of the algorithms.

6 Conclusion

Our proposed algorithm MT* is substantially faster than the state-of-the-art algorithm to solve the multi-robot LTL optimal motion planning problem. Computation times for MT* increases linearly with increase in workspace size. We were able to generate motion plan till 8 robots for 30×30 sized workspace whereas the state-of-the-art algorithm hardly scales up to 3 robots over 15×15 sized workspace. In our future work, we will extend our algorithm to deal with dynamic obstacles and preferential constraints.

References

Baier, C., and Katoen, J.-P. 2008. *Principles of Model Checking (Representation and Mind Series)*. The MIT Press.

Balch, T., and Arkin, R. 1998. Behavior-based formation control for multirobot teams. *IEEE Transaction on Robotics and Automation* 14(6):926–939.

Bhatia, A.; Kavraki, L. E.; and Vardi, M. Y. 2010a. Sampling-based motion planning with temporal goals. In *2010 IEEE International Conference on Robotics and Automation*, 2689–2696.

Bhatia, A.; Kavraki, L. E.; and Vardi, M. Y. 2010b. Motion planning with hybrid dynamics and temporal goals. In *CDC*, 1108–1115.

Chen, Y.; Tůmová, J.; and Belta, C. 2012. LTL robot motion control based on automata learning of environmental dynamics. In *ICRA*, 5177–5182.

Cormen, T. H.; Leiserson, C. E.; Rivest, R. L.; and Stein, C. 2009. *Introduction to Algorithms, Third Edition*. The MIT Press, 3rd edition.

Duret-Lutz, A., and Poitrenaud, D. 2004. Spot: An extensible model checking library using transition-based generalized büchi automata. In *MASCOTS*.

Fox, D.; Burgard, W.; Kruppa, H.; and Thrun, S. 2000. A probabilistic approach to collaborative multi-robot localization. *Autonomous Robots* 8(3):325–344.

Halperin, D.; Latombe, J.-C.; and Wilson, R. H. 1998. A general framework for assembly planning: The motion space approach. In *Annual Symposium on Computational Geometry*, 9–18.

Hart, P. E.; Nilsson, N. J.; and Raphael, B. 1968. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics* 4(2):100–107.

Jennings, J. S.; Whelan, G.; and Evans, W. F. 1997. Cooperative search and rescue with a team of mobile robots. In *ICRA*, 193–200.

Kantaros, Y., and Zavlanos, M. M. 2017. Sampling-based control synthesis for multi-robot systems under global temporal specifications. In *Proceedings of the 8th International Conference on Cyber-Physical Systems*, ICCPS '17, 3–13. New York, NY, USA: ACM.

Karaman, S., and Frazzoli, E. 2009. Sampling-based motion planning with deterministic μ-calculus specifications. In *CDC*, 2222–2229.

Khalidi, D., and Saha, I. 2018. T* : A heuristic search based algorithm for motion planning with temporal goals. *CoRR* abs/1809.05817.

Kress-Gazit, H.; Fainekos, G. E.; and Pappas, G. J. 2007. Where's Waldo? Sensor-based temporal logic motion planning. In *ICRA*, 3116–3121.

LaValle, S. M., and James J. Kuffner, J. 2001. Randomized kinodynamic planning. *The International Journal of Robotics Research* 20(5):378–400.

LaValle, S. M. 2006. *Planning Algorithms*. New York, NY, USA: Cambridge University Press.

Plaku, E., and Karaman, S. 2016. Motion planning with temporal-logic specifications: Progress and challenges. *AI Commun.* 29(1):151–162.

Rodríguez, S., and Amato, N. M. 2010. Behavior-based evacuation planning. In *ICRA*, 350–355.

Rus, D.; Donald, B.; and Jennings, J. 1995. Moving furniture with teams of autonomous robots. In *IROS*, 235–242.

Russell, S., and Norvig, P. 2009. *Artificial Intelligence: A Modern Approach*. Upper Saddle River, NJ, USA: Prentice Hall Press, 3rd edition.

Saha, I.; Ramaithitima, R.; Kumar, V.; Pappas, G. J.; and Seshia, S. A. 2014. Automated composition of motion primitives for multi-robot systems from safe LTL specifications. In *IROS*, 1525–1532.

Smith, S. L.; Tůmová, J.; Belta, C.; and Rus, D. 2010. Optimal path planning under temporal logic constraints. In *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 3288–3293.

Ulusoy, A.; Smith, S. L.; Ding, X. C.; Belta, C.; and Rus, D. 2013. Optimality and robustness in multi-robot path planning with temporal logic constraints. *I. J. Robotic Res.* 32(8):889–911.

Vasile, C. I., and Belta, C. 2013. Sampling-based temporal logic path planning. *CoRR* abs/1307.7263.

Wongpiromsarn, T.; Topcu, U.; and Murray, R. M. 2012. Receding horizon temporal logic planning. *IEEE Trans. Automat. Contr.* 57(11):2817–2830.