MT*: Multi-Robot Path Planning for Temporal Logic Specifications*

Dhaval Gujarathi[1] and Indranil Saha[2]

***Abstract*— We address the path planning problem for a team of robots satisfying a complex high-level mission specification given in the form of a Linear Temporal Logic (LTL) formula. 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 that captures the collective motion of the robots and then computation of the shortest path using Dijkstra's shortest path algorithm. We propose MT*, an algorithm that reduces the computation burden for generating such 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 scales well with both the number of robots and the size of the workspace.**

I. Introduction

Path 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 [1]. A basic path planning task involves point-to-point navigation while avoiding obstacles and satisfying some user-given constraints. Recently, there has been an increased interest in specifying complex paths for robots using Linear Temporal Logic (LTL) (e.g. [2], [3], [4], [5], [**?**], [6], [7], [8], [9], [10]). Using temporal logic [11], one can specify requirements that involve a temporal relationship between different operations performed by robots.

This paper focuses on the class of multi-robot LTL path planning problems where the inputs are the *discrete dynamics* of the robots and a global LTL specification. Though we deal with any specification that can be captured in LTL, our main focus is to deal with those specifications that require the robots to repeat some tasks perpetually. Such requirements arise in many robotic applications, including persistent surveillance [6], search and rescue [12], assembly planning [13], evacuation [14], localization [15], object transportation [16], and formation control [17].

Traditionally, the LTL path 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 [18] is employed to generate an optimal trajectory satisfying an LTL query [19], [6]. However, for a large workspace and a complex LTL specification, this approach is merely scalable. We seek to design a computationally efficient algorithm to generate optimal trajectories for the robots.

Heuristic-based search algorithms such as A* (for a single robot) [20] and M* (for a multi-robot system) [21] have been successfully applied to solving point-to-point path planning problems and are proven to be significantly faster than Dijkstra's shortest path algorithm. Heuristic search-based algorithms have also been applied to solving the temporal logic path planning problem for a single robot [22], [23], [24], [25], [9]. In this paper, we introduce the MT* algorithm that, for the first time, attempts to incorporate the heuristic search in generating an *optimal* trajectory for a multi-robot system satisfying a global LTL query efficiently. We apply our algorithm to solving various LTL path planning problems for a multi-robot system in 2-D workspaces and compare the results with that of the algorithm presented in [6]. Our experimental results demonstrate that MT* in many cases achieves an order of magnitude better computation time than that of the traditional approach [6] to solve the optimal LTL path planning problem.

Related Work. MT* is a special class of multi-agent path finding (MAPF) problems. MAPF is a widely studied planning problem where the goal is to find collision-free paths for a number of agents from their initial locations to some specified goal locations [26], [27]. MAPF is a special class of the finite LTL path synthesis problem [28] for multi-agent systems where the specification of each robot is given by the LTL formula "eventually $goal$", where $goal$ is the proposition that becomes `true` when all the robots reach their goal locations. A number of previous works addressed the multi-robot path planning problem for general finite LTL specifications [7], [29], [30].

On the contrary, our focus in the paper is to address the planning problem for those LTL specifications that capture perpetual behavior and is satisfied using infinite trajectories, like the work presented in [31], [32], [6], [33], [34], [35], [36]. Among the above-mentioned work, Kloetzer et al. [31] and Shoukry et al. [34] solve the problem without discretizing the robot dynamics. However, such techniques are computationally demanding and scale poorly with the number of robots. In all the other papers, the robot dynamics are discretized into weighted transition systems. The work by Tumova et al. [33] assumes that the robots are assigned their own tasks, with some communication requirements among the robots. In this paper, we deal with a single specification for the multi-robot system, where finding the optimal distribution of the tasks is the core challenge. Chen et al. [32] divide the problem by first decomposing the given specification into the specifications for the individual robots and then generating the plans for the robots based on their own specification. This decoupling leads to suboptimality as a given LTL specification may have a number of valid decompositions, and their quality can be known only after the plans are generated. Recent work by Kantaros and Zavlanos

*The authors thankfully acknowledge the Defence Research & Development Organisation (DRDO), India for funding the project through JCBCAT, Kolkata.

[1]Dhaval Gujarathi is with the SAP Labs, India. This work was carried out when Dhaval was an M.Tech student at the Department of Computer Science and Engineering, IIT Kanpur, India. dhavalsgujarathi@gmail.com

[2]Indranil Saha is with the Department of Computer Science and Engineering, IIT Kanpur, India isaha@cse.iitk.ac.in

to solve the LTL path planning problem scales for a large number of robots [35], [36]. They employ the sampling-based technique to compute the first feasible trajectory, which might not be cost-optimal. Unlike their work, we focus on generating trajectories with minimum cost.

II. Problem

A. Preliminaries

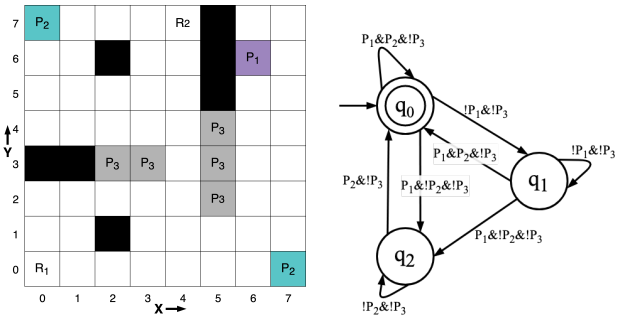
1) Workspace, Robot Actions and Trajectory: We assume that a team of n robots operates in a 2-D or a 3-D discrete workspace $\mathcal{W}$ which we represent as a grid map. The grid divides the workspace into square-shaped cells. Every cell in the workspace $\mathcal{W}$ is referenced using its coordinates. Some cells can be marked as obstacles and cannot be visited by any robot. We denote the 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 the costs of the actions to generate the trajectory.

2) Transition System: We model the motion of the robot i in the workspace $\mathcal{W}$ as a weighted transition system defined as $T^i := (S^i, s^i_0, E^i, \Pi^i, L^i, w^i)$, where (i) S^i is the set of states denoting the obstacle-free cells in $\mathcal{W}$, (ii) $s^i_0 \in S^i$ 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, $(s^i_1, s^i_2) \in E^i$ iff $s^i_1, s^i_2 \in S^i$ and $s^i_1 \xrightarrow{act} s^i_2$, 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.

3) Joint Transition System: A joint transition system T is a transition system that 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) $s_0 := \langle s^1_0, s^2_0, ..., s^n_0 \rangle \in S_T$ 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 $(s^i_1, s^i_2) \in E^i$ 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 $L_T(s_j) := \bigcup_{i=1}^{n} L^i(s^i_j)$ gives us set of propositions true at state s_j, (vi) $w_T : E^T \rightarrow \mathbb{N}_{>0}$, and $w_T(s_1, s_2) := \sum_{i=1}^{n} w^i(s^i_1, s^i_2)$ is a weight function.

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

Example **1.** *Throughout this paper, we will use a surveillance example for illustration purposes. The workspace $\mathcal{W}$ is shown in Figure 1(a). We build a transition systems T^i for all the robots over $\mathcal{W}$ where $\Pi^i = \Pi_T = \{P_1, P_2, P_3\}$. Throughout this paper, we will use a surveillance example for illustration purposes. The workspace $\mathcal{W}$ is shown in Figure 1(a). We build a transition systems T^i for all the robots over $\mathcal{W}$ where $\Pi^i = \Pi_T = \{P_1, P_2, P_3\}$.*



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

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

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

The proposition P_1 is satisfied if the robot visits the location $(6, 6)$, whereas the proposition P_2 can be satisfied by visiting either $(0, 7)$ or $(7, 0)$. The proposition P_3 corresponds to the locations that are prohibited for the robots to visit. Cells with black color represent obstacles ($\mathcal{O}$). We assume that from any cell in $\mathcal{W}$, a robot can move to one of its four neighboring cells with a cost of 1 or stay at the same location with a cost of 0. □

4) Linear Temporal Logic: The path planning query/task in our work is given in terms of formulae written using *Linear Temporal Logic* (LTL). LTL formulae over the set of atomic propositions Π_T are formed according to the following grammar [11]:

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

The basic ingredients of an LTL formula are the atomic propositions $a \in \Pi_T$, the Boolean connectors like $\wedge$ (conjunction) and $\neg$ (negation), 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 ϕ, if the first state of σ satisfies ϕ. The logical operators $\wedge$ and $\neg$ have their usual meaning. For an LTL formula ϕ, $\mathrm{X}\phi$ is `true` in a state if ϕ is satisfied at the next step. The formula $\phi_1 \,\mathrm{U}\, \phi_2$ 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 $\Box$ (*Always*) and $\Diamond$ (*Eventually*). The formula $\Box\phi$ denotes that ϕ must be satisfied all the time in the future. The formula $\Diamond\phi$ denotes that ϕ has to hold sometime in the future. We have denoted negation $\neg P$ as $!P$ and conjunction as $\&$ in the Figures.

5) Büchi Automaton: For any LTL formula ϕ over a set of propositions Π_T, we can construct a Büchi automaton with input alphabet $\Pi_B = 2^{\Pi_T}$ that captures all the possible ways in which the given task ϕ can be completed [37]. 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, (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. A final 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.

Example **2.** *Figure 1(b) shows the Büchi automaton for an LTL task $\Box(\Diamond P_1 \wedge \Diamond P_2 \wedge \neg P_3)$, which means that the robots should always repeat visiting locations corresponding to proposition P_1 and P_2, and always avoid locations corre-*

sponding to proposition P_3. Here, q_0 is the start state as well as the final state. It informally depicts the steps to be followed to complete the task ϕ. For example, the transitions $q_1 \rightarrow q_2$ leads the robot to visit a state where $P_1 \wedge \neg P_2 \wedge \neg P_3$ is satisfied (only P_1 is satisfied, but P_2 and P_3 are not satisfied) by going through only those states which satisfy $\neg P_1 \wedge \neg P_3$. This way, we can also understand the meaning of the other transitions. □

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 [19] for examples.

B. Problem Definition

Consider a team of robots moving in a static workspace $\mathcal{W}$ represented as transition systems $\{T_1, \ldots, T_n\}$ 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. A final state signifies the completion of task ϕ. Our objective is to find an infinite length path $\mathcal{R} = s_0, s_1, s_2, \ldots$ over T which satisfies ϕ and so, there exists $f \in F_\pi$ which occurs on $\mathcal{R}$ infinitely many times. Such path can be divided into two components namely *prefix* ($\mathcal{R}_{pre}$) and *suffix* ($\mathcal{R}_{suf}$) [11]. A prefix is a finite run from the initial state s_0 to a final 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. The suffix is repeated periodically and infinitely many times to generate an infinite length run $\mathcal{R}$. Thus, we can represent run $\mathcal{R}$ as $\mathcal{R}_{pre}.\mathcal{R}_{suf}^{\omega}$, where $\mathcal{R}_{pre} = s_0, s_1, s_2, \ldots, s_p$ be a prefix and $\mathcal{R}_{suf} = s_{p+1}, s_{p+2}, \ldots, s_{p+r}$ be a suffix, $s_{p+r} = s_p$ and ω denotes the suffix being repeated infinitely many times.

The cost of such run can be minimized if we minimize the cost of the suffix, which can be given as

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

PROBLEM **1.** *Given a joint transition system T capturing the motion of the team of robots in workspace $\mathcal{W}$ and an LTL formula 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* (1).

Note: If we want to deal with the LTL specifications that can be satisfied by the finite trajectories (for example, $\Diamond a$ or $a_1 U a_2$), we can define a cost function that captures the cost of the prefix, denoted by $\mathcal{C}(\mathcal{R}_{pre})$.

C. Baseline Solution Approach

The state-of-the-art solution to the above problem [6] uses the automata-theoretic model checking approach. It computes the 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 (see the full version of the paper [38] for details). 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 a hindrance to scaling up this algorithm. In the next section, we present our algorithm MT* and use the algorithm proposed in [6] as the baseline for quantitative comparison.

III. MT* ALGORITHM

In MT*, we divide a complex LTL path planning problem into simpler problems systematically, which can be solved individually and then combined to solve the original problem optimally. MT* only computes a reduced version of the product graph P, which we call the *Abstract Reduced Graph* G_r. Its size is significantly smaller compared to P and thus is faster to compute and consumes less memory.

A. 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 $\Box(\Diamond P_1 \wedge \Diamond P_2 \wedge \neg P_3)$ whose Büchi automaton B is shown in Figure 1(b). Let T^1 be the transition system of the robot constructed from $\mathcal{W}$. For one robot system, the joint transition system T is the same as T^1. Consider a product automaton P of T and B. Suppose $s_0 = \langle(4,7)\rangle$ and therefore $S_{P,0} = (\langle(4,7)\rangle, q_0)$. From here, we must use the transitions in the 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 $(\langle(4,6)\rangle, q_1)$ which satisfies $\neg P_1 \wedge \neg P_3$ from $(\langle(4,7)\rangle, q_0)$ as per the definition of the product automaton. 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 $\neg P_1 \wedge \neg P_3$ formula on the self-loop on q_1. Suppose we next move from $(\langle(4,6)\rangle, q_1)$ to $(\langle(6,6)\rangle, q_2)$ on P which satisfies $P_1 \wedge \neg P_2 \wedge \neg P_3$ and this path is $(\langle(4,6)\rangle, q_1) \rightarrow (\langle(4,5)\rangle, q_1) \rightarrow (\langle(4,4)\rangle, q_1) \rightarrow \ldots \rightarrow (\langle(6,5)\rangle, q_1) \rightarrow (\langle(6,6)\rangle, q_2)$. On the path from $(\langle(4,6)\rangle, q_1)$ to $(\langle(6,6)\rangle, q_2)$, all the intermediate nodes satisfy the self-loop transition condition on q_1. We can consider the self-loop transition condition $\neg P_1 \wedge \neg P_3$ over q_1 as a constraint that must be satisfied by the intermediate states while completing a task of moving to the location satisfying the transition condition from q_1 to q_2. 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 $(\langle(6,6)\rangle, q_2)$ in the reduced graph assuming that there exists a path between these two states. We explore this path opportunistically only when it is required. This is the first idea behind MT*.* □

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.

Definition 1 (Negative and Positive Transition Condition)**.** *A transition condition which is a conjunction of all negative propositions is called a* negative transition condition *and is*

Algorithm 1: MT*

Input: Transition systems $\{T^1, ..., T^n\}$, ϕ: An LTL formula
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_Neighbors`(c_{pos})
end
$G_r(S_r, v_0, E_r, F_r, \mathcal{N}) \leftarrow$ `Generate_Redc_Graph`$(\mathtt{B}, \mathtt{T}, \mathtt{S}_*)$
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)
    end
    $\mathcal{R}_f^{suf}(\langle \mathcal{R}_s^1, .., \mathcal{R}_s^n \rangle) \leftarrow$ `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)
  end
end
$\mathcal{R}_P^{suf} \leftarrow \underset{\mathcal{R}_f^{suf} \text{ with a valid prefix}}{\operatorname{argmin}} \mathcal{C}\left(\mathcal{R}_f^{suf}\right)$
$\mathcal{R}_P^{pre} \leftarrow$ 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, S_*)
  $v_{init} \leftarrow v_0(s_0, q_0)$
  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)
        end
      end
    end
    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)
        end
      end
      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)
        end
      end
    end
  end
  return G_r

denoted by c_{neg}. The one which is not negative is called a positive transition condition*, and is denoted by c_{pos}.*

EXAMPLE **4.** *$\neg P_1 \wedge \neg P_3$ is a negative whereas $P_1 \wedge \neg P_2 \wedge \neg P_3$ is a positive transition condition.* □

Before we formally define abstract reduced graph, we define the abstract state, abstract neighbor set, and the edges in the abstract reduced graph.

Definition 2 (Abstract state)**.** *An abstract state in the abstract reduced graph is a pair of a multi-robot state and a Büchi automaton state where in the multi-robot state, some robot states may be unknown. 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 use a map L_* to store the tasks and the constraints that each robot satisfies in an abstract state.*

Definition 3 (Abstract Neighbor Set)**.** *For a given positive transition condition c_{pos}, the abstract neighbor set is defined as the set of all possible abstract states of the multi-robot system that can be reached after executing the transition. We represent it as $S_*[c_{pos}]$.*

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 denoting the locations for robots who do not receive any task from c_{pos} as $(*, *)$.

EXAMPLE **5.** *Consider that $c_{pos} = P_2 \wedge \neg P_3$ be a transition condition from B. Suppose here we are planning the paths for two robots. Thus, to satisfy this transition, one of the robots must go to a location where $P_2 \wedge \neg P_3$ is satisfied, and at the same time, the other robot must be at a location where $\neg P_3$ is satisfied. There are 2 locations $(0, 7)$ or $(7, 0)$ which satisfy $P_2 \wedge \neg P_3$ and 52 locations which satisfy $\neg P_3$. So, there are overall $2 \cdot 2 \cdot 52 = 208$ ways in which both robots can collectively satisfy $P_2 \wedge \neg P_3$ and we represent these 208 satisfying configurations symbolically as $S_*[c_{pos}] = \{\langle (0, 7), (*, *), q_0 \rangle, \langle (7, 0), (*, *), q_0 \rangle, \langle (*, *), (0, 7), q_0 \rangle, \langle (*, *), (7, 0), q_0 \rangle\}$. Here, $\langle (0, 7), (*, *), q_0 \rangle$ says that robot 1 is at location $(7, 0)$ whereas robot 2 could be at any obstacle-free location which satisfies $\neg P_3$. Here, $L_*[\langle (0, 7), (*, *), q_0 \rangle] = \{\{P_2, \neg P_3\}, \{\neg P_3\}\}$ and so on.* □

Using the above ideas, we represent the product graph symbolically as a significantly smaller abstract reduced graph.

Definition 4 (Edge in Abstract Reduced Graph)**.** *An edge from node $v_i(s_i, q_i)$ to some node $v_l(s_l, q_l)$ in the abstract reduced graph is added under two conditions stated below.*

(a) Distant Neighbor Condition: If $\exists \delta_B(q_i, c_{neg}) = q_i$ 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 any other state q_j in the Büchi automaton, 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, we do not add nodes using 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. Here, q_i and q_l can be same.

(b) Product Automaton Condition: If the distant neighbor condition is not satisfied, 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 $s_l \in S_*[c_{pos}]$. For all c_{neg} outgoing transitions from q_i to some q_l, we add an

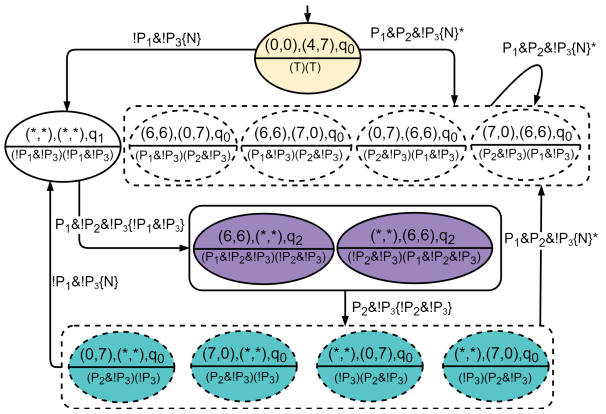


Fig. 2: Abstract Reduced Graph for a 2 robot system having the workspace and the Büchi automaton from Figure 1

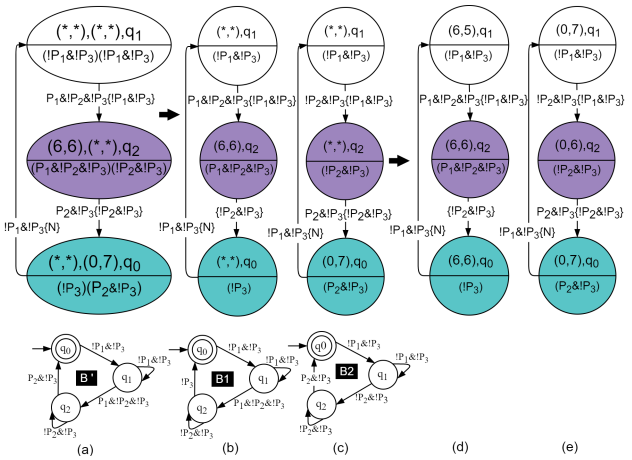


Fig. 3: Suffix computation in MT* algorithm

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 same. At this moment, we do not know the complete value of the nodes in s_i and s_l.

We use $\mathcal{N}$ to track neighbor information in the abstract reduced graph. For an edge (v_a, v_b), $\mathcal{N}(v_a, v_b) =$ `true` says that v_b must be a neighbor of v_a in T. On the other hand, $\mathcal{N}(v_i, v_l) =$ `false` says that v_i and v_l are not neighbors (connected by an edge) in the product graph P.

EXAMPLE **6.** *In Figure 2, the edge between $v_i = (\langle(6,6)(*,*)\rangle, q_2)$ and $v_l = (\langle(7,0)(*,*)\rangle, q_0)$ is added using the distant neighbor condition. Here, the path between $(6,6)$ and $(7,0)$ could be established through c_{neg} transition condition. Similarly, whatever value we fill for unknowns $(*,*)$, an intermediate path between the two must satisfy c_{neg} condition. On the other hand, the edge between $v_i = (\langle(*,*)(7,0)\rangle, q_0)$ and $v_l = (\langle(6,6)(7,0)\rangle, q_0)$ is added using the product automaton condition. Here, the location of robot 2 does not change. Robot 1's location in v_i is not known (denoted by $(*,*)$) and is $(6,6)$ in v_l. So, later in the algorithm, whenever we fill this unknown value, we have to ensure that it is a neighbor of $(6,6)$ in T*[1]. □

Now we provide the formal definition of Abstract Reduced Graph.

Definition 5 (Abstract Reduced Graph)**.** *Given a joint transition system T and a Büchi automaton B, the Abstract Reduced Graph is defined as $G_r := (S_r, v_{init}, E_r, F_r)$, where $v_{init} = (s_0, q_0)$ is an initial state, S_r and E_r are the set of vertices and edges respectively added using the distant neighbor condition and product automaton condition by running the Breadth-First-Search (BFS) algorithm starting from node (s_0, q_0). $F_r \subseteq S_r$ denotes the set of final states. For $f \in F_r$, the Büchi Automaton state component q_f of f is a final state of the Büchi Automaton B, i.e., $q_f \in Q_f$.*

EXAMPLE **7.** *The abstract reduced graph generated for a two robot system over the workspace $\mathcal{W}$ and the Büchi automaton B from Figure 1 is shown in Figure 2. All the nodes enclosed in a rectangle represent an abstract neighbor set S_* for some transition. Edges have transition conditions written on them. We mention N on the transition to indicate that the $\mathcal{N}$ value for that transition is* `true`*. The * on the transition says that this transition is applicable only if the nodes are actual neighbors. The transition condition accompanied with c_{neg} condition in curly braces $\{\}$ represents a c_{neg} transition that must be used to reach the next node. In every node below the robot coordinates, we show the L_* value for that node.*

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 dequeued from the queue Q, so we add the neighbors 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 neighbors of v_0. Here, there exists a transition from 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 neighbors to v_0. However, none of these nodes are neighbors 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 infeasibility we represent using $*$ on the transition condition. Now, there also exists a transition from q_0 to q_1 with transition condition $\neg P_1 \wedge \neg P_3$. 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 the product automaton condition, so whatever value we choose to put in place of $\langle (*,*)(*,*) \rangle$ must be a neighbor of v_0. Now, suppose $v_i = (\langle (*,*), (*,*) \rangle, q_1)$ has been dequeued from the queue Q. Here, there exists a negative 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 neighbors 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. As these nodes have been added as neighbors to v_i using the distant neighbor condition, they may not be actual neighbors of v_i in P. We explain this in the next section. Like this, we add nodes to G_r. The complete G_r is shown in Figure 2. □

B. MT* Procedure

We outline all the steps of MT* in Algorithm 1 and explain the concepts for the major steps in detail. Given the transition systems for n robots $\{T^1, ..., T^n\}$ and an LTL query ϕ, the goal is to compute a minimum cost run $\mathcal{R}^i$ over T^i for n robots, satisfying ϕ in the form of prefix and suffix.

We first compute the Büchi automaton from the given LTL task ϕ. Then for all c_{pos} transition conditions present in B, we compute the abstract neighbor set $S_*[c_{pos}]$ using the procedure `Abstract_Distant_Neighbor()` as explained in section (III-A). Then we generate Abstract Reduced Graph (G_r) using procedure `Generate_Redc_Graph()` from Algorithm 1. Now, for each state $f \in F_r$, we compute all the possible simple cycles (cycles containing no other cycle) starting and ending at f. We choose the one with the minimum cost and reachable from v_{init} as our final solution. We can easily find such cycles using Depth-First Search (DFS) algorithm starting with node f.

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 the example mentioned in Figure 3 for a better understanding of the readers. (i) `Extract_Buchi_Trans_From_cf( )`: 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 and 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 sub-graph 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_Cf_Over_Ti( )`: 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_Cfi_Over_B'( )`: Using this procedure, we derive n automata $\{B^1, ..., B^n\}$ from the transitions of cycles $\{c_f^1, ..., c_f^n\}$. These automata represent the path/constraints that each robot must follow in this particular joint trajectory/task C_f. Automata B^1 and B^2 are shown in Figure 3(b) and Figure 3(c). (v) `Optimal_Run( )`: In this procedure, we complete the individual incomplete trajectories c_f^i for all the robots using their individual automaton B^i. For example, the incomplete trajectory shown in Sub-figure (b) is completed using automaton 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 automaton B^1 in T^i using single robot LTL pathfinding algorithms T* [9] or `Optimal_Run` [19]. In these algorithms, as we now have a 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 system. 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-the-art algorithm [6].

(vii) `Sync( )`: When we generate the robot trajectories independently for individual robots, the generated trajectories can be of different lengths, and thus, the sequence in which particular locations are visited may change and may not satisfy ϕ. There are two types of transitions in G_r. One is added using the product automaton condition, in which both the nodes should be neighbors. In this case, all the generated individual trajectories will have a consistent Büchi state. The second one is the transition added using the distant neighbor 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 lengths. However, if the robots can be stopped at some location during their operation, it is straightforward to achieve synchronization by keeping the robot waiting at the initial location of the transition for a sufficient duration. A hypothetical example demonstrating this is shown in the full version of the paper [38].

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 to sum of individual costs which is $2+2=4$.

If, for a robot, all the nodes in the abstract individual trajectory are $(*,*)$, then this robot is not doing anything in this team task sequence C_f, and thus can be ignored. 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 $(*,*)$.

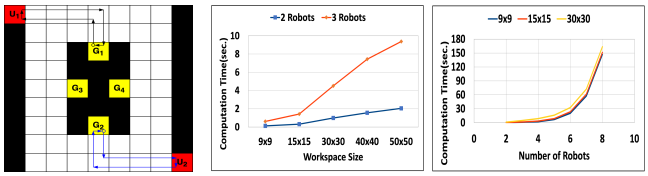
`Compt_Prefix( )`: After synchronizing the individual trajectories, we now know the exact coordinates of the final state $f \in F_r$ in C_f. For example, C_f in 3(a) has $f = (\langle(*,*)(0,7)\rangle, q_0)$ which we compute as $(\langle(6,6)(0,7)\rangle, q_0)$. Now, in this step, we can find a prefix path from the initial state $v_{init} = (s_0, q_0)$ to this computed f using the same steps we used to compute the suffix (we find a path instead of a 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. From all those valid suffix cycles, we choose the one with the minimum cost as our final outcome and project it over $\{T^1, .., T^n\}$ 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 $(\langle(6,6)\rangle, q_2) \xrightarrow{\neg P_2 \wedge \neg P_3} (\langle(*,*)\rangle, q_0) \xrightarrow{\neg P_1 \wedge \neg P_3 \{N\}} (\langle(*,*)\rangle, q_1) \xrightarrow{P_1 \wedge \neg P_2 \wedge \neg P_3 \{\neg P_1 \wedge \neg P_3\}} (\langle(6,6)\rangle, q_2)$.

Correctness and Optimality. The proof of optimality and correctness of MT* is available in the full version of the paper [38], where we prove the following theorem.

Theorem 1. *The trajectory $\mathcal{R}_T$ computed by MT* algorithm satisfies the given LTL formula ϕ, and it is the minimum cost trajectory among all the trajectories satisfying ϕ.*

Complexity. The computation time of MT* increases exponentially with the increase in the number of robots and the size of the LTL specification as these increase the size of the abstract reduced graph and thus the number of cycles to be explored. However, as the size of the abstract reduced graph remains the same with an increase in the size of



(a) Trajectories for ϕ_2 (b) Computation Time(s) Vs Workspace Size (c) Computation Time(s) Vs No. of Robots

Fig. 4: Results for LTL Query ϕ_2

the workspace, the computation time increases polynomially with the increase in the workspace size due to the polynomial increase in the computation time to find the distance between two locations of interest. This provides a significant advantage over the baseline algorithm. As the precise complexity analysis of MT* is complex, we rely on the experimental evaluation to demonstrate its efficacy.

IV. Evaluation

In this section, we present several results to establish the computational efficiency of MT* algorithm against the baseline solution [6]. We implement our algorithm in C++. We use LTL2TGBA tool [39] as the LTL query to Büchi automaton converter. The results have been obtained on a desktop computer with a 3.4 GHz quadcore processor with 16 GB of RAM. The C++ implementations of MT* algorithm and the baseline algorithm are available in the following repository: `https://github.com/iitkcpslab/MTStar`.

We use the 2-D workspace shown in Figure 4(a) (borrowed from [6]). Each grid cell has 4 neighbors. The cost of each edge between the neighboring cells is 1 unit. In the workspace, U_1 and U_2 are data upload locations, whereas G_1, G_2, G_3, and G_4 are data gather locations.

We evaluate MT* algorithm for five LTL queries $\phi_1, \phi_2, \ldots, \phi_5$ borrowed from [6]. We define propositions over the workspace in the following way: $gather$: Data is gathered from a gather station, $rXgather$: Robot X gathers data from a gather station, $gatherY$: Data is gathered from gather station Y, $rXgatherY$: Robot X gathers data from gather station Y. We define the propositions for 'upload' in the same way. Due to space constraints, we discuss only one of the five queries in this section. Detailed results for all the five queries are available in the full version of the paper [38].

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

$$\begin{aligned}\phi_2 = \Box\Diamond gather \wedge \Box(gather \Rightarrow (r1gather \wedge r2gather)) \\ \wedge \Box(r1gather \Rightarrow \mathrm{X}(\neg(r1gather)\mathrm{U}(r1upload))) \\ \wedge \Box(r2gather \Rightarrow \mathrm{X}(\neg(r2gather)\mathrm{U}(r2upload)))\end{aligned}$$

The generated trajectory for ϕ_2 is shown in Figure 4(a) 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 $\Box(gather \Rightarrow (r1gather \wedge r2gather))$. They upload it to the nearest upload station to minimize the total cost and move again to $G1$ and $G2$ respectively, to complete the cyclic trajectory. The total cost of the multi-robot trajectory for the robots is 24.

In Table I, we list down the computation times of the baseline solution and MT*, and the speed-up achieved by MT* over the baseline solution for queries $\phi_1, \ldots, \phi_5$. Büchi states column lists the number of states in the Büchi automaton for the corresponding LTL mission. We list these results for different sizes of workspaces. A 9×9 workspace is shown in Figure 4(a). The 15×15 and 30×30 workspaces are similar to the 9×9 map with the same number of data gather and data upload locations. In the table, we can observe a significant speed up that MT* achieves over the baseline solution. 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.

For the graphs in Figure 4, we have used workspaces from size 9×9 till 50×50. In Figure 4(b), we observe the performance of MT* with the increase in workspace size for 2 and 3 robot systems 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` faces polynomial increase with the increase in the workspace size, and thus MT* achieves a polynomial increase in the computation time with the increase in the workspace size. In Figure 4(c), we observe that the computation time of MT* increases exponentially with the increase in the number of robots for query ϕ_2. This is because, with the increase in the number of robots, the number of possible task assignments increases exponentially. These results are consistent for other queries and workspaces.

In Table II, we compare the number of vertices and edges in Product Graph P with Abstract Reduced Graph G_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. It is significantly smaller than the product graph, and that is why it has superior performance 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.

A video simulation of a ground robot for two different specifications has been submitted as a supplementary material and is also available at `https://youtu.be/3iAhycehys8`.

V. Discussions

Our proposed algorithm MT* is substantially faster than the state-of-the-art algorithm to solve the multi-robot LTL optimal path planning problem. MT* does not attempt to provide collision-free trajectories unless the requirement of collision avoidance is explicitly specified in the input LTL formula. Thus, MT* is useful for high-level strategic planning for a temporal logic specification. We assume that, during the actual execution of the plans, some dynamic real-time collision avoidance algorithms such as the ones presented in [40], [41], [42] will be employed to ensure collision avoidance among the robots.

In MT*, we evaluate the cost of the cycles containing a final state one by one while keeping track of the minimum cost cycle. Once we complete the computation of the first cycle, we have a valid solution (a trajectory satisfying the LTL specification). In the subsequent iterations, we look for a more optimal solution. Thus, MT* is a good candidate for

TABLE I: Baseline Solution (B.S.) Vs MT*

		9 × 9 Workspace							15 × 15 Workspace				30 × 30 Workspace			
		2 Robots			3 Robots			8 Robots	2 Robots			8 Robots	2 Robots			8 Robots
ϕ	Büchi States	B.S. (s)	MT* (s)	Speed Up	B.S. (s)	MT* (s)	Speed Up	MT* (s)	B.S. (s)	MT* (s)	Speed Up	MT* (s)	B.S. (s)	MT* (s)	Speed Up	MT* (s)
ϕ_1	12	20.60	13.60	**1.50**	–	266.00	–	5397.19	3635	19.80	**184**	5513.59	–	37.41	–	6022.69
ϕ_2	5	2.80	0.10	**21.90**	9786	0.61	**16037**	147.11	74	0.30	**243**	151.54	985	1.00	**985**	163.47
ϕ_3	5	22.30	0.90	**23.70**	11665	5.50	**2107**	1211.12	496	2.50	**198**	1196.55	9043	8.30	**1086**	1311.95
ϕ_4	5	1.80	0.04	**40.0**	1150	0.10	**11500**	66.36	38	0.06	**664**	66.48	728	0.12	**6129**	67.53
ϕ_5	5	11.40	0.05	**218.8**	–	0.19	–	382.33	579	0.07	**7846**	383.66	10937	0.14	**80633**	383.99

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

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

an *anytime* implementation (like anytime A* [43]). Moreover, it is possible to parallelize the evaluation of the suffix cycles (for different final states) in MT* to boost the performance further.

REFERENCES

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

[2] H. Kress-Gazit, G. E. Fainekos, and G. J. Pappas, "Where's Waldo? Sensor-based temporal logic motion planning," in *ICRA*, 2007, pp. 3116–3121.

[3] S. Karaman and E. Frazzoli, "Sampling-based motion planning with deterministic μ-calculus specifications," in *CDC*, 2009, pp. 2222–2229.

[4] A. Bhatia, L. E. Kavraki, and M. Y. Vardi, "Sampling-based motion planning with temporal goals," in *ICRA*, 2010, pp. 2689–2696.

[5] T. Wongpiromsarn, U. Topcu, and R. M. Murray, "Receding horizon temporal logic planning," *IEEE Transaction on Automatic Control*, vol. 57, no. 11, pp. 2817–2830, 2012.

[6] A. Ulusoy, S. L. Smith, X. C. Ding, C. Belta, and D. Rus, "Optimality and robustness in multi-robot path planning with temporal logic constraints," *I. J. Robotic Res.*, vol. 32, no. 8, pp. 889–911, 2013.

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

[8] T. Kundu and I. Saha, "Energy-aware temporal logic motion planning for mobile robots," in *ICRA*, 2019, pp. 8599–8605.

[9] D. Khalidi, D. Gujarathi, and I. Saha, "T* : A heuristic search based algorithm for motion planning with temporal goals," in *ICRA*, 2020.

[10] P. Purohit and I. Saha, "DT*: Temporal logic path planning in a dynamic environment," in *IROS*, 2021, pp. 3627–3634.

[11] C. Baier and J.-P. Katoen, *Principles of Model Checking*. MIT Press, 2008.

[12] J. S. Jennings, G. Whelan, and W. F. Evans, "Cooperative search and rescue with a team of mobile robots," in *ICRA*, 1997, pp. 193–200.

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

[14] S. Rodríguez and N. M. Amato, "Behavior-based evacuation planning," in *ICRA*, 2010, pp. 350–355.

[15] D. Fox, W. Burgard, H. Kruppa, and S. Thrun, "A probabilistic approach to collaborative multi-robot localization," *Autonomous Robots*, vol. 8, no. 3, pp. 325–344, 2000.

[16] D. Rus, B. Donald, and J. Jennings, "Moving furniture with teams of autonomous robots," in *IROS*, 1995, pp. 235–242.

[17] T. Balch and R. Arkin, "Behavior-based formation control for multi-robot teams," *IEEE Transaction on Robotics and Automation*, vol. 14, no. 6, pp. 926–939, 1998.

[18] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 3rd ed. The MIT Press, 2009.

[19] S. L. Smith, J. Tůmová, C. Belta, and D. Rus, "Optimal path planning under temporal logic constraints," in *IROS*, 2010, pp. 3288–3293.

[20] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.

[21] G. Wagner and H. Choset, "Subdimensional expansion for multirobot path planning," *Artificial Intelligence*, vol. 219, pp. 1–24, 2015.

[22] F. Bacchus and F. Kabanza, "Planning for temporally extended goals," *Ann. Math. Artif. Intell.*, vol. 22, no. 1-2, pp. 5–27, 1998.

[23] J. A. Baier and S. A. McIlraith, "Planning with first-order temporally extended goals using heuristic search," in *AAAI*, 2006, pp. 788–795.

[24] ——, "Planning with temporally extended goals using heuristic search," in *ICAPS*, 2006, pp. 342–345.

[25] F. Patrizi, N. Lipovetzky, G. De Giacomo, and H. Geffner, "Computing infinite plans for LTL goals using a classical planner," in *IJCAI*, 2011, pp. 2003–2008.

[26] H. Ma and S. Koenig, "AI buzzwords explained: multi-agent path finding (MAPF)," *AI Matters*, vol. 3, no. 3, pp. 15–19, 2017.

[27] I. Saha, R. Ramaithitima, V. Kumar, G. J. Pappas, and S. A. Seshia, "Implan: Scalable incremental motion planning for multi-robot systems," in *ICCPS*, 2016, pp. 43:1–43:10.

[28] A. Camacho, J. A. Baier, C. J. Muise, and S. A. McIlraith, "Finite LTL synthesis as planning," in *ICAPS*, 2018, pp. 29–38.

[29] P. Schillinger, M. Bürger, and D. V. Dimarogonas, "Decomposition of finite LTL specifications for efficient multi-agent planning," in *DARS*, 2016, pp. 253–267.

[30] ——, "Multi-objective search for optimal multi-robot planning with finite LTL specifications and resource constraints," in *ICRA*, 2017, pp. 768–774.

[31] M. Kloetzer and C. Belta, "Automatic deployment of distributed teams of robots from temporal logic motion specifications," *IEEE Transaction on Robotics*, vol. 26, no. 1, pp. 48–61, 2010.

[32] Y. Chen, X. C. Ding, A. Stefanescu, and C. Belta, "Formal approach to the deployment of distributed robotic teams," *IEEE Transaction on Robotics*, vol. 28, no. 1, pp. 158–171, 2012.

[33] J. Tumova and D. V. Dimarogonas, "Multi-agent planning under local LTL specifications and event-based synchronization," *Automatica*, vol. 70, pp. 239–248, 2016.

[34] Y. Shoukry, P. Nuzzo, A. Balkan, I. Saha, A. L. Sangiovanni-Vincentelli, S. A. Seshia, G. J. Pappas, and P. Tabuada, "Linear temporal logic motion planning for teams of underactuated robots using satisfiability modulo convex programming," in *CDC*, 2017, pp. 1132–1137.

[35] Y. Kantaros and M. M. Zavlanos, "Sampling-based optimal control synthesis for multirobot systems under global temporal tasks," *IEEE Transaction on Automatic Control*, vol. 64, no. 5, pp. 1916–1931, 2019.

[36] ——, "Stylus*: A temporal logic optimal control synthesis algorithm for large-scale multi-robot systems," *Int. J. Robotics Res.*, vol. 39, no. 7, 2020.

[37] P. Gastin and D. Oddoux, "Fast LTL to Büchi automata translation," in *CAV*, 2001, pp. 53–65.

[38] D. Gujarathi and I. Saha, "MT* : Multi-robot path planning for temporal logic specifications," *CoRR*, vol. abs/2103.02821, 2021. [Online]. Available: https://arxiv.org/abs/2103.02821

[39] A. Duret-Lutz and D. Poitrenaud, "SPOT: An extensible model checking library using transition-based generalized büchi automata," in *MASCOTS*, 2004, pp. 76–83.

[40] J. van den Berg, S. J. Guy, M. C. Lin, and D. Manocha, "Reciprocal *n*-body collision avoidance," in *ISRR*, 2009, pp. 3–19.

[41] D. Hennes, D. Claes, W. Meeussen, and K. Tuyls, "Multi-robot collision avoidance with localization uncertainty," in *AAMAS*, 2012, pp. 147–154.

[42] A. Best, S. Narang, and D. Manocha, "Real-time reciprocal collision avoidance with elliptical agents," in *ICRA*, 2016, pp. 298–305.

[43] M. Likhachev, G. J. Gordon, and S. Thrun, "ARA*: Anytime A* with provable bounds on sub-optimality," in *NIPS*, 2003, pp. 767–774.