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

Dhaval Gujarathi
SAP Labs, India
Email: dhavalsgujarathi@gmail.com

Indranil Saha
IIT Kanpur, India
Email: isaha@cse.iitk.ac.in

Abstract—We address the path planning problem for a team of robots satisfying a complex high-level mission specification given in the form of an 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 which 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 unlike the state of the art approach, 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 [28]. 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 the robots using temporal logic (e.g. [27, 23, 9, 10, 48, 14, 44, 37, 24]). Using temporal logic [2], one can specify requirements that involve a temporal relationship between different operations performed by robots.

This paper focuses on the class of multi-robot Linear Temporal Logic (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 [44], assembly planning [18], evacuation [34], search and rescue [20], localization [17], object transportation [55], and formation control [7].

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 [15] is employed to generate an optimal trajectory satisfying an LTL query [42, 44]. 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) [36] and M* (for a multi-robot system) [47] 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 [11, 4, 5, 33, 24]. 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 an 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 [44]. Our experimental results demonstrate that MT* in many cases achieves an order of magnitude better computation time than that of the traditional approach [44] to solve the optimal LTL path planning problem.

Related Work. MT* is a special class of multi-agent path finding (MAPF) problem. 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 [32, 58]. MAPF is a special class of the finite LTL path synthesis problem [11] 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 [37, 39, 40].

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 [25, 13, 44, 43, 41, 21, 22]. Among the above-mentioned work, Kloetzer et al. [25] and Shoukry et al. [41] 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. [43] assumes that the robots are assigned their own tasks, with
some communication requirement 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. [13] 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 Kantaras and Zavlanos to solve the LTL path planning problem scales for a large number of robots [21]. They employ 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 $W$ which we represent as a grid map. The grid divides the workspace into square-shaped cells. Every cell in the workspace $W$ 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 $O$. 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) Transition System: We model the motion of the robot $i$ in the workspace $W$ 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, (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 $s^i_1 \not\equiv act s^i_2$, where $act \in Act$, (iv) $\Pi^i$ 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) $w^i : E^i \rightarrow N_{>0}$ 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 $W$, 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 $(s_1, s_2, ..., s_n)$, $s^i$ represents the state of robot $i$ in transition system $T^i$, (ii) $s_0 := (s^1_0, s^2_0, ..., s^n_0) \in S_T$ 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 $(s_1, s_2) \in E^i$ 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 $L_T(s_1) := \bigcup_{i=1}^n L^i(s_1)$ gives us set of propositions true at state $s_1$, (vi) $w_T : E_T \rightarrow N_{>0}$, and $w_T(s_1, s_2) := \sum_{i=1}^n w^i(s_1, s_2)$ 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 warehouse pick-and-drop example for illustration purposes. The workspace $W$ is shown in Figure 1(a). We build a transition systems $T$ for all the robots over $W$ 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 $P_i$. Here, $P_1$, $P_2$, and $P_3$ correspond to drop location, pickup location, and the locations to be avoided by the robots, respectively. Cells with black colour represent obstacles ($O$). We assume that from any cell in $W$, a robot can move to one of its four neighbouring cells with cost 1 or stay at the same location with cost 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 $\Pi_T$ are formed according to the following grammar (2):

$$\Phi ::= \text{true} \mid \alpha \mid \phi_1 \land \phi_2 \mid \neg \phi \mid X\phi \mid \phi_1 \text{ U } \phi_2$$

The basic ingredients of an LTL formula are the atomic propositions $\alpha \in \Pi_T$, the Boolean connectors like conjunction $\land$ and negation $\neg$, and two temporal operators $X$ (next) and $U$ (until). The semantics of an LTL formula is defined over an infinite trajectory $\sigma$. The trajectory $\sigma$ satisfies a formula $\xi$, if the first state of $\sigma$ satisfies $\xi$. The logical operators conjunction $\land$ and negation $\neg$ have their usual meaning. For an LTL formula $\phi$, $X\phi$ is true in a state if $\phi$ is satisfied at the next step. The formula $\phi_1 \text{ U } \phi_2$ denotes that $\phi_1$ must remain true until $\phi_2$ 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 $\phi$ must be satisfied all the time in the future. The formula $\Diamond \phi$ denotes that $\phi$ has to hold sometime in the future. We have denoted negation $\neg P$ as $\neg P$ and conjunction as $\&$ in the Figures.
5) Büchi Automaton: For any LTL formula $\phi$ over a set of propositions $\Pi_T$, 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 $\Pi_B$ to get accepted.

**Example 2.** Figure 7(b) shows the Büchi automaton for an LTL task $\square(\Diamond P_2 \land \Diamond P_2 \land \neg P_3)$, which means that the robots should always repeat visiting pick up locations $P_2$ and drop location $P_1$, and always avoid locations $P_3$. Here, $q_0$ is the start state as well as the final state. It informally depicts the steps to be followed in order to complete the task $\phi$. The transitions $q_1 \rightarrow q_2 \rightarrow q_0$ lead us to visit a state where $P_1 \land \neg P_2 \land \neg P_3$ is satisfied by going through only states which satisfy $\neg P_1 \land \neg P_3$ and then go to state where $P_2 \land \neg P_3$ is satisfied using states which satisfy $\neg P_2 \land \neg P_3$. 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 $P := (S_B, S_B \times Q_B, E_P, F_P, W_P)$, where (i) $S_B = S_T \times Q_B$, (ii) $S_B := (s_0, q_0)$ is an initial state, (iii) $E_P \subseteq S_B \times S_P$, where $((s_i, q_k), (s_j, q_l)) \in E_P$ if and only if $(s_i, s_j) \in E_T$ and $(q_k, \ell_T(s_j), q_l) \in \delta_B$, (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 [42] for examples.

**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 $\phi$ over $\Pi_T$ which needs to be completed collectively by them repetitively and infinitely many times. We construct a Büchi automaton $B$ from $\phi$. Let $\Pi_c = \{c | c \in \Pi_B \land \exists \delta_B(q_i, c) = q_j \}$ where, $q_i \in Q_B$ and $q_j \in Q_f$. Let $F_\pi = \{s_i | s_i \in E_T \land s_i \ni \pi_j \land \pi_j \in \Pi_c\}$. $F_\pi$ represents a set of all the possible final states (final state is the last state in the complete trajectory in $T$ which satisfies $\phi$) 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 $F_\pi$.

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

$$C(R) = \lim_{i \rightarrow +\infty} \sum_{k=t^I_\ell(i)}^{t^I_\ell(i) + 1} w_T(s_k, s_{k+1})$$

(1)

1) Prefix-Suffix Structure: The accepting run $R$ of infinite length can be divided into two components namely prefix ($R_{\text{pre}}$) and suffix ($R_{\text{ suf}}$). A prefix is a finite run from the 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 $R$. So, we can represent run $R$ as $R_{\text{pre}} \cdot R_{\text{ suf}}$, where $\omega$ denotes the suffix being repeated infinitely many times.

**Lemma 3.1:** For every run $R$ which satisfies the LTL formula $\phi$ and minimizes cost function (1), there exists a run $R_c$ which satisfies $\phi$, minimizes cost function (1) and is in prefix-suffix structure. Refer [42] for the proof.

The cost of such run $R_c$ 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 $R = R_{\text{pre}} \cdot R_{\text{ suf}}^\omega$, where $R_{\text{pre}} = s_0, s_1, s_2, \ldots, s_p$ be a prefix and $R_{\text{ suf}} = s_{p+1}, s_{p+2}, \ldots, s_{p+r}$ be a suffix, where $s_{p+r} = s_p$. We can redefine the cost function given in (1) as

$$C(R) = C(R_{\text{ suf}}) = \sum_{i=p+1}^{p+r-1} w_T(s_i, s_{i+1})$$

(2)

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

**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 $C(R_{\text{pre}})$.

**C. Baseline Solution Approach**

The baseline solution to above problem uses the automata-theoretic model checking approach [44], the steps of which are outlined in the Algorithm [44].

The first step in this algorithm is to compute the joint transition system $T$ from the transition systems of the individual robots $T_i$. Then we compute the Büchi automaton from the given LTL query $\phi$. We then compute the product automaton of $T$ and $B$. In this product automaton, for each final state $f \in F_\pi$, we find a prefix run starting from initial state $S_{P,0}$ to $f \in F_\pi$ and then find minimum cost cycle starting and ending at $f$ using Dijkstra’s algorithm. We then choose the prefix-suffix pair with the smallest $C(R_p)$ cost i.e. the pair with smallest suffix cost, and project it on $T$ to obtain the run $R_T$.
which represents the joint motion all the robots in $W$. We then project $\mathcal{R}_T$ over individual $T^i$ to obtain the $i$-th joint robot $\mathcal{R}^i$. The trajectory $\mathcal{R}^i$ for the $i$-th robot provides us with the cyclic trajectory which the robot can follow repetitively to complete the given task $\phi$ repetitively.

In the following section, we present $MT^*$ algorithm that provides a significantly improved running time for generating an optimal trajectory satisfying a given LTL query.

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 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 $W$ shown in Figure 1(a) and has been given an LTL task $\square((P_1 \land \diamond P_2 \land \neg P_3))$ whose Büchi automaton $B$ is shown in Figure 1(b). Let $T^1$ be the transition system of the robot constructed from $W$. For one robot system, the joint transition system $T$ will be same as $T^1$. Consider a product automaton $P$ of $T$ and $B$. Suppose $s_0 = ((4,7))$ and therefore $P_{s_0} = ((4,7),q_0)$. Now, 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 $((4,6),q_1)$ from $((4,7),q_0)$ as per the definition of the product automaton. From $((4,6),q_1)$, we must visit a location where $P_1 \land \neg P_2 \land \neg P_3$ is satisfied so that we can move to Büchi state $q_0$ from $q_1$. All the intermediate states till we reach such a state must satisfy $P_1 \land \neg P_3$ formula. Suppose we next move from $((4,6),q_1)$ to $((6,6),q_2)$ on $P$ which satisfies $P_1 \land \neg P_2 \land \neg P_3$ and this path is $((4,6),q_1) \rightarrow ((4,5),q_1) \rightarrow ((4,4),q_1) \rightarrow \ldots \rightarrow ((6,5),q_1) \rightarrow ((6,6),q_2)$. On the path from $((4,6),q_1)$ to $((6,6),q_2)$.

Algorithm 2: MT$^*$

1. **Input**: Transition systems $\{T^1, \ldots, T^n\}$, $\phi$: An LTL formula
2. **Output**: A run $(R_1, \ldots, R^n)$ that satisfies $\phi$
3. $B(Q_B,q_0,\Pi_B,\delta_B,Q_f) \rightarrow \text{ltl_to_Buchi}(\phi)$
4. for all $q_i,q_j \in Q_B$, where $\delta_B(q_i,c_{pos}) = q_j$
5. $S_r[c_{pos}] = \text{Abstract_Distant_Neighbours}(c_{pos})$
6. $G_r(s_v,v_0,E_v,F_v,N) \rightarrow \text{Generate_Redc_Graph}(B,T)$
7. for all $f \in F$,
8. for each simple cycle $C_f$ containing $f$ in $G_r$
9. $R^f \rightarrow \text{Extract_Buchi_Trans_From_C_(C_f,B)}$
10. for $i \rightarrow \{1, \ldots, n\}$
11. $c_i \rightarrow \text{Project_C_(C_i, G_r)}$
12. $B' \rightarrow \text{Project_C_(C_i, B)}$
13. $R^f \rightarrow \text{Optimal_Run}(B', c_i, T)$
14. $R^f \rightarrow \text{Compute_Prefix}(f, B, G_r)$
15. $\text{argmin}_{R^f} C(R^f)$
16. $\text{argmax}_{\text{prefix}} C(R^f)$
17. $\text{prefix}(f)$
18. $R_p = \text{Redc_r}(R^f)$
19. $\text{Redc_g}(G_r, T)$
20. $\text{Redc_g}(G_r, T)$
21. $\text{Redc_g}(G_r, T)$
22. $\text{Redc_g}(G_r, T)$
23. $\text{Redc_g}(G_r, T)$
24. $\text{Redc_g}(G_r, T)$
25. $\text{Redc_g}(G_r, T)$
26. $\text{Redc_g}(G_r, T)$
27. $\text{Redc_g}(G_r, T)$
28. $\text{Redc_g}(G_r, T)$
29. $\text{Redc_g}(G_r, T)$
30. $\text{Redc_g}(G_r, T)$
31. $\text{Redc_g}(G_r, T)$
32. $\text{Redc_g}(G_r, T)$
33. $\text{Redc_g}(G_r, T)$
34. $\text{Redc_g}(G_r, T)$
35. $\text{Redc_g}(G_r, T)$
36. $\text{Redc_g}(G_r, T)$
37. $\text{Redc_g}(G_r, T)$
38. $\text{Redc_g}(G_r, T)$
39. $\text{Redc_g}(G_r, T)$
40. $\text{Redc_g}(G_r, T)$
41. $\text{Redc_g}(G_r, T)$
42. $\text{Redc_g}(G_r, T)$
43. $\text{Redc_g}(G_r, T)$
44. $\text{Redc_g}(G_r, T)$
45. $\text{Redc_g}(G_r, T)$
46. $\text{Redc_g}(G_r, T)$
47. $\text{Redc_g}(G_r, T)$
48. $\text{Redc_g}(G_r, T)$
49. $\text{Redc_g}(G_r, T)$
50. $\text{Redc_g}(G_r, T)$

all the intermediate nodes satisfy the self-loop transition condition on \( q_1 \). We can consider the self-loop transition condition \( \neg P_1 \land \neg P_3 \) over \( q_1 \) as the constraint which 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 \(((\{4,6\},q_1)\) to state \(((\{6,6\},q_2)\) 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* 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, \( P_1 \land \neg P_3 \) is a negative whereas \( P_1 \land \neg P_2 \land \neg P_3 \) is a positive transition condition. For any transition, we can consider all the positive propositions as the 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 \( P_1 \land P_2 \land \neg P_3 \), some robot must visit a location where \( P_3 \) 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 \( P_1 \land P_2 \land \neg P_3 \) completely and collectively. Let us move to the second idea.

**Example 4.** Consider that \( c_{pos} = P_2 \land \neg P_3 \) be a transition condition from \( B \). Suppose here we are planning the paths for two robots. So, to satisfy this transition, one of the robots must go to a location where \( P_2 \land \neg P_3 \) is satisfied. There are two ways to achieve it. The first one is when robot 1 reaches a location on \( W \) where \( P_2 \land \neg P_3 \) is satisfied, and robot 2 is at the location where \( \neg P_3 \) is satisfied. Then we can say that \( P_2 \land \neg P_3 \) satisfied by both the robots. And second one is when robot 2 satisfies \( P_2 \land \neg P_3 \) and robot 1 satisfies \( \neg P_3 \). In the first case, robot 1 could be at either \((0,7)\) or \((7,0)\) which satisfies \( P_2 \land \neg P_3 \) and robot 2 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 is \( 52 \times 2 = 104 \). Similarly, in another 104 ways, the robots can satisfy the second case. We represent all these \( 104 + 104 = 208 \) satisfying configurations symbolically as \( \{(0,7),(*,*)\}, \{(7,0),(*,*)\}, \{(*,0),(0,7),(*,*)\} \). 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 \( S_s[c_{pos}] \). 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 \( s_* := (s_1^*,...,s_n^*) \). We can easily compute \( S_s[c_{pos}] \) 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 \((*,*)\). While computing \( S_s \), we also store the tasks and the constraints that each robot satisfies in a particular configuration using a map \( L_s \). For example, \( S_s[c_{pos} = P_2 \land P_3] = \{(0,7),(*,*)\}, \{(7,0),(*,*)\}, \{(*,0),(0,7),(*,*)\} \) and \( L_s([[0,7],[*,*]) = \{\{P_2,\neg P_3\}, \{\neg P_3\}\} \) and so on.

Using the above ideas, we can represent the product graph symbolically as a significantly smaller 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 the following rules:

- **Condition:** \( \exists \delta_B(q_i,c_{neg}) = q_i \) and \( \exists \delta_B(q_i,c_{neg}) = q_j \) where, \( q_i \neq q_j \), 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.

- **If Condition is true then** add an edge from \( v_i(s_i,q_i) \) to all \( v_j(s_j,q_j) \) such that \( \exists \delta_B(q_i,c_{pos}) = q_j \) and \( s_i \in S_s[c_{pos}] \). Here, \( q_i \) and \( q_j \) 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 \( v_j \) later in the algorithm. We use \( \bar{N} \) to keep track neighbour information in \( G_r \), \( \bar{N}(v_i,v_j) = \text{true} \) says that \( v_i \) must be a neighbour of \( v_j \) in \( T \). In this condition, we set \( \bar{N}(v_i,v_j) = \text{false} \). It says that \( v_i \) and \( v_j \) may not be actual neighbours in the actual product graph \( P \). For example, consider an edge between \( v_i = (((6,6),(*,*)\),q_2) \) and \( v_j = (((7,0),(*,*)\),q_0) \) in Figure 3. 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.

- **If Condition is false then** the same as above, we add an edge from \( v_i \) to all \( v_j(s_j,q_j) \) such that \( \exists \delta_B(q_i,c_{pos}) = q_j \) and \( s_j \in S_s[c_{pos}] \). And for all \( c_{neg} \) outgoing transitions from \( q_j \) to some \( q_k \), we add an edge from \( v_i \) to \( v_m(s_i,q_i) \). Here, \( s_i := (s_i^1, ..., s_i^n) \) where, \( s_i^1 = (*,*) \). Here, \( q_i \) and \( q_k \) can be the same. We call this way of adding neighbours as product automaton way/condition. For all these conditions, we set \( \bar{N}(v_i,v_j) \) to \text{true} which says that \( v_i \) and \( v_j \) are actual neighbours in \( T \). At this moment we do not know the complete value of the nodes in \( v_i \) and \( v_j \). For example, as shown in Figure 3, \( v_i \) could be\(((*,),(7,0)),(q_0)\) and \( v_j \) could be\(((6,6),(7,0)),(q_0)\). Here for robot 2, \((7,0)\) from \( v_i \) is the neighbour of \((7,0)\) form \( v_j \). But, robot 1’s location in \( v_i \) is not known (represented as \((*,*)\)) and is known in \( v_j \) 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 \( T \). We formally define the Abstract Reduced Graph for the transition system \( T \) and the Büchi automaton \( B \) as \( G_r := (S_r,v_{init},E_r,F_r) \), 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) \( E_r \subseteq S \times S_r \), is a set of edges added as per the above conditions, (iv) the set of final states \( F_r \subseteq S \times S_r \).
We have shown the 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 q0 to q1 with transition condition ¬P1 ∧ ¬P3. And as this is a c_neq type transition, we add an edge from v0 to (((*,*),(*,*)),q1). Again this node has been added as per product automaton condition, so whatever value we choose to put in place of (((*,*),(*,*))) must be neighbour of v0.

Now, suppose v1 = (((*,*),(*,*)),q1) has been de-queued from the queue Q. Here, there exists a negative self loop over q1 with transition condition c_neq = ¬P1 ∧ ¬P3 and there does not exist any c_neq transition from q1 to any other state in B. So, we add all the nodes as neighbours to v1 which satisfy c_pos transition conditions and ignore c_neq self loop. There exists a positive transition from q1 to q2 with transition condition c_pos = P1 ∧ ¬P2 ∧ ¬P3. So, we add edges from v1 to all the nodes in S_e[(P1 ∧ ¬P2 ∧ ¬P3)] which are ((6,6),(6,6)),(((6,6),(6,6)),((0,7),(6,6)), and (7,0),(6,6)) with Büchi state q2. As these nodes have been added as neighbours to v1 using distant neighbour condition, they may not be actual neighbours of v1 in P. We explain this in depth in the next section. Like this, we add nodes to G_r. Completed G_r is shown in Figure 2.

B. MT* Procedure

We outline all the block steps of MT* in Algorithm 2 and explain the concepts for the major steps. Given the transition systems for n robots \{T^1,...,T^n\} and an LTL query \phi, the goal is to compute a minimum cost run R^i over T^i for n robots, satisfying \phi in the form of prefix and suffix.

(1) In this algorithm, we first compute the Büchi automaton from the given LTL task \phi. (2) Then for all c_pos transition conditions present in B, we compute the abstract neighbour set S_e[c_pos] using the procedure Abstract_Distant_Neighbour() as explained in section III-A. (3) Then we generate Abstract Reduced Graph (G_r) using procedure Generate_Redc_Graph() from Algorithm 2. (4) Final state f_p ∈ 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 that satisfies LTL task $\phi$, 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 the suffix. We can always obtain a suffix with a smaller cost by removing this extra cycle. So, we only search for a simple cycle (in which no node is repeated except the starting and the ending node $f$). We can easily find such cycles using Depth-First Search (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 the example mentioned in Figure 3(a) for a better understanding of the readers. (i) Extract_Buchi_Trans From_c_c( ): 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 $f = (((*,*),(0,7)),q_0)$ 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_T'( ): We use this procedure to project/extract $C_f$ over $T'$ to compute trajectory $c_f'$ 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_Cf_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 (c). (v) Optimal_Run( ): In this procedure, we complete the individual incomplete trajectories $c_f'$ 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 $s_{source}^1 = (((6,6),q_0)$. Then we find the next known node on the cyclic trajectory which is also $s_{next}^1 = (((6,6),q_0)$. Then we attempt to find the path from $s_{source}^1$ to $s_{next}^1$ with automaton $B^1$ in $T'$ using single robot LTL pathfinding algorithms $T'$ [24] or Optimal_Run [42]. 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 art algorithm [44]. (vii) Sync( ): When we generate the robot trajectories independently for individual robots, the generated trajectories may not be in sync. 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 $\phi$. However, if the robots can be stopped at some location during their operation, it is straightforward to achieve synchronization. There are two types of transitions in $G_r$. One is added using the product automaton condition, in which both the nodes should be neighbours. In this case, all the generated individual trajectories will have a consistent Büchi state. The 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 lengths.

**EXAMPLE 6.** Consider an example of some hypothetical LTL query in which there is an edge from $(((6,6),(6,6)),q_1) \xrightarrow{P_2 \wedge P_3} (((7,0),(7,0)),q_2)$ 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 $(((6,6),q_1)$ to $(((7,0),q_2)$ for robot 1 is $p_1 = (((6,6)),q_1) \xrightarrow{P_2 \wedge P_3} (((7,6),q_1) \xrightarrow{P_2 \wedge P_3} ((7,7),q_1)$ (4 nodes) $\neg P_2 \wedge \neg P_3 \xrightarrow{(((7,1),q_1) \rightarrow P_2 \wedge P_3 \xrightarrow{(((7,0),q_2)$ consisting of total 6 transitions with $c_{neg}$ transition condition and one $c_{pos}$ transition ($P_2 \wedge \neg P_3$). Whereas, for robot 2, individual trajectory is $p_2 = (((6,6),q_1) \xrightarrow{P_2 \wedge P_3} (((6,5),q_1) \xrightarrow{P_2 \wedge P_3}((6,6),q_1) \xrightarrow{P_2 \wedge P_3}(((1,7),q_1) \rightarrow P_2 \wedge P_3 \xrightarrow{(((0,7),q_2)$ 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 $(((6,6),q_1)$ in $p_1$ (shorter trajectory) to extend it as $p_1' = (((6,6),q_1) \xrightarrow{P_2 \wedge P_3} (((6,6),q_1) \xrightarrow{P_2 \wedge P_3} ... (9 nodes) ... \neg P_2 \wedge \neg P_3) \rightarrow (((6,6),q_1) \rightarrow P_2 \wedge P_3 \xrightarrow{(((7,6),q_1) \rightarrow P_2 \wedge P_3 \rightarrow (((7,1),q_1) \rightarrow P_2 \wedge P_3 \rightarrow (((7,0),q_2)$ to make both the individual trajectories synchronized to have same no. of transitions for same transition condition.

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 $C_f$. In such a case, we can directly ignore this robot. For example, consider a $C_f = (((7,0),(*,*)),q_0) \rightarrow (((*,*),(*,*)),q_1) \rightarrow (((6,6),(*,*)),q_2)$ 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 $R^{syn}_{n^2f} = (((6,5),(0,7)),q_1) \rightarrow (((6,6),(0,6)),q_2) \rightarrow (((6,6),(0,7)),q_0) \rightarrow (((6,5),(0,7)),q_1)$ with cost equal to sum of individual costs which is $2 + 2 = 4$. (viii) 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 Figure 3(a) has $f =$
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 \( \{ R^1, \ldots, R^n \} \). For the Abstract Reduced Graph \( G_r \) shown in Figure 4, the suffix cycle with the minimum cost is \( C_f = ((6,6), (0,7)), q_0 \rightarrow ((6,6), (0,7)), q_0) \) 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 \( ((6,6), q_2) \rightarrow ((*,*), q_1) \rightarrow \neg P_1 \land \neg P_2 \land P_3 \rightarrow ((*,*), q_1) \rightarrow \neg P_1 \land \neg P_2 \land P_3 \rightarrow ((*,*), q_1) \rightarrow ((6,6), q_2) \).

**Correctness and Optimality.** In this section, we prove the correctness of \( MT^* \) algorithm. To prove the correctness of \( MT^* \), we will have to show that the suffix run which we find in the algorithm satisfies the given LTL formula, and it is the minimum cost suffix run among all the satisfying runs.

**Theorem 1.** The suffix run \( R_{suf} \) computed by \( MT^* \) algorithm follows the given LTL formula \( \phi \) and it is the minimum cost run among all the \( \phi \) satisfying runs.

**Proof:** In \( MT^* \) algorithm, we work on the Abstract Reduced Graph \( G_r \), which is a reduced version of the Product Graph \( P \). First, we will have to prove that Abstract Reduced Graph preserves all the minimum cost paths starting and ending a state \( f_p \in F_P \).

**Lemma 1.** Abstract Reduced Graph preserves all the minimum cost paths starting and ending at a state \( f_p \in F_P \).

**Proof:** First, we claim that all the final states \( F_P \) present in \( P \) are preserved in \( F_r \). All the incoming transitions to the final Büchi automaton states are of type \( c_{pos} \). This is because whenever we specify some task in the form of an LTL query, it contains at least one positive proposition (positive propositions represent actual task whereas negative propositions specify constraints to be followed by the robots). And LTL task with only constraints, i.e., negative propositions will be meaningless in the context of robotic applications) and as Büchi final state signifies the completion of the given task, the incoming transition to Büchi final state will always be of type \( c_{pos} \). In \( G_r \), we add all the nodes which satisfy \( c_{pos} \) transitions, and these nodes are in abstract form. So, from this, we can say that we add all the nodes from set \( F_P \) in abstract form to \( G_r \). We denote \( F_P \) in abstract form as \( F_r \). In Abstract Reduced Graph, we add nodes using two conditions. First is the product graph condition, in which added node should be neighbour. In this condition, we added \((*,*)\) as neighbour. We can always substitute any neighbouring node in place of \((*,*)\). So no transition is lost for the nodes added using this condition. The second one is the distant neighbour condition, in which we mean to use \( c_{neg} \) type self-loop to establish a path. We use Dijkstra’s to establish this path. In distant neighbour condition, we lose some transition as we skip transitions due to \( c_{neg} \) self-loop. But, we can use Dijkstra’s algorithm with \( c_{neg} \) constraint and recover the shortest path between the nodes which were added using the distant neighbour condition. So, from these arguments, we can say that Abstract Reduced Graph preserves the minimum cost paths starting and ending at a state \( f_p \in F_P \).

Now, consider a simple cycle \( C_f \) in Abstract Reduced Graph. It represents a possible task assignment for the robots. We then compute the trajectories for individual robots such that collectively they follow constraints in \( C_f \), and each individual robot is computed using either Dijkstra’s algorithm or \( A^* \) algorithm. So, the overall moving cost of the robots is minimized. In \( MT^* \), we repeat this procedure for all the possible cycles. So all the possible task assignments are considered.

So, from these arguments, we conclude that the suffix run \( R_{suf} \) computed by \( MT^* \) algorithm follows the given LTL formula \( \phi \) and it is the minimum cost run among all the \( \phi \) satisfying runs.

**Complexity.** 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 the workspace, the computation time increases linearly with the increase in the workspace size due to the linear increase in the distance between two locations of interest. This provides a significant advantage over the baseline algorithm. As the precise complexity analysis \( MT^* \) 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 \([44]\). The results have been obtained on a desktop computer with a 3.4GHz quadcore processor with 16GB of RAM. We use LTL2TGBA tool \([16]\) as the LTL query to Büchi automaton converter. The C++ implementations of \( MT^* \) and
Baseline algorithms and a simulation video are submitted as supplementary materials. We use the 2-D workspace as shown in Figure 4(a) (borrowed from [44]). Each grid-cell has 4 neighbours. The cost of each edge between the neighbouring cells is 1 unit. In the workspace, U1 and U2 are data upload locations 1 and 2, whereas G1, G2, G3 and G4 are data gather locations 1 to 4.

We have evaluated MT* algorithm for five LTL queries $\phi_1, \phi_2, \ldots, \phi_5$ borrowed from [44]. We define propositions over the workspace in the following way: gather: Data has been gathered from a gather station, $rX\text{g}ather$: Robot X has gathered data from a gather station, gatherY: Data has been gathered from the gather station Y, $rX\text{g}atherY$: Robot X has gathered data from the gather station Y. We define propositions for ‘upload’ in the same way.

**Query $\phi_1$:** The mission is “Repeatedly gather data from data gather locations and once you gather the data, upload it to data upload location before gathering new data.”

$$\phi_1 = \Box \Diamond \text{gather \land}$$

$$\Box (r_1\text{gather} \implies X(\neg(r_1\text{gather}) \cup (r_1\text{upload}))) \land$$

$$\Box (r_2\text{gather} \implies X(\neg(r_2\text{gather}) \cup (r_2\text{upload})))$$

This task induces a sequence among the locations to be visited. Here, Once a robot gathers data at any data gathering station, it must visit a data upload station before it can gather the data again. It also induces a response(visit upload station) to an event(visit to gather station) in the robot trajectory. The trajectory generated by this specification is shown in Figure 5(a). We can see that only robot 1 circling between G1 and F1 and robot 2 remains at its initial location. This happened because we never said in the specification that both the robots should gather data. We only asked to gather data, and a trajectory is generated based on that. Robot 2 remains at its location, as the cost of not moving is 0, and we are trying to minimize the cost. The cost of this trajectory is 12 (number of movements).

**Query $\phi_2$:** The 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.”

$$\phi_2 = \Box (\text{gather} \land$$

$$\Box (r_1\text{gather} \implies X(\neg(r_1\text{gather}) \cup (r_1\text{upload}))) \land$$

$$\Box (r_2\text{gather} \implies X(\neg(r_2\text{gather}) \cup (r_2\text{upload}))) \land$$

$$\Box (\text{gather} \implies (r_1\text{gather} \land r_2\text{gather}))$$

The newly added constraint asks both the robots to gather the data and that too at the same time. These kinds of constraints can be used to induce synchronization among the robots. The trajectories for this specification are shown in Figure 5(b). Here, both the robots gather data at the same time and upload it before visiting the data gather station again. In Figure 5(b), a small circle on the robot trajectory denotes the starting point of the cycle. The cost of the trajectories generated for this specification is 24.

**Query $\phi_3$:** The mission is “Each Robot must repeatedly visit data gather location at same time synchronously to gather data but not to the same gather station and upload that data at upload station before gathering the new data again.”

$$\phi_3 = \phi_2 \land \Box (\neg(r_1\text{gather1} \land r_2\text{gather1}) \land$$

$$\neg(r_1\text{gather2} \land r_2\text{gather2}) \land$$

$$\neg(r_1\text{gather3} \land r_2\text{gather3}) \land$$

$$\neg(r_1\text{gather4} \land r_2\text{gather4}))$$

In case 2, the robots were asked to gather data at the same time. One of the possible trajectories is shown in Figure 5(c), in which both the robots gather data from the same gather station and upload it to the same upload station. In case 3, we ask the robots to gather data from the different gather stations. So, Robot 1 gathers data from G1 and Robot 2 from G2. This way, they follow the specification and also minimize the cost of the movement. This kind of specification can be used to avoid repetitive work. The trajectories generated for this specification are shown in Figure 5(c). We can also observe that the data gathering task has been synchronized but at different data gathering locations. The cost for this run is 24.
Query $\phi_4$: The mission is “Robot 1 must repeatedly visit data gather location $G_3$ and robot 2 must repeatedly visit data gather location $G_2$ at the same time to gather data and upload that data at upload station before gathering the data again.”

$$\phi_4 = \Box \Diamond \text{gather} \land$$
$$\Box (r_1\text{gather} \implies X(\neg (r_1\text{gather}) \lor (r_1\text{upload}))) \land$$
$$\Box (r_2\text{gather} \implies X(\neg (r_2\text{gather}) \lor (r_2\text{upload}))) \land$$
$$\Box (\text{gather} \implies ((r_1\text{gather}) \land (r_2\text{gather})))$$

In this specification, we specifically assign the data gather location to each of the robots, and they have to choose the closest data upload station to upload that data before gathering more data. The trajectories for both the robots are shown in Figure 5(d). The total cost of the run is 26. This kind of specification can be used to assign a certain task to a specific robot from a team of robots.

Query $\phi_5$: The mission is “Gather data from all the gather stations.”

$$\phi_5 = \Box \Diamond \text{gather1} \land \Box \Diamond \text{gather2} \land \Box \Diamond \text{gather3} \land$$
$$\Box \Diamond \text{gather4}$$

In this case, we are only interested in gathering data. But data must be gathered from all the gather stations. Here, we just repeatedly want to gather data from all the gather stations. The robot trajectories are shown in Figure 5(e). Here, robot 1 is covering three of the four gather stations, and robot 2 just stays at the gather station 4. This happened because this kind of trajectory has the minimum cost, i.e., this is the best possible task assignment possible. The cost of this run is 26.

In Table II, we list down the computation times of the baseline solution and $MT^*$, and the speeded-up achieved by $MT^*$ over the baseline solution for queries $\phi_1, \ldots, \phi_5$. B"uchi states column lists the number of states in the B"uchi automaton for the corresponding LTL mission. We list these results for different sizes of workspaces. A $9 \times 9$ workspace is shown in Figure 4(a). The $15 \times 15$ and $30 \times 30$ workspaces are similar to the $9 \times 9$ map with the same number of data gather and data upload locations. In the table, we can observe 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 due to very high computation time and memory requirement.

For the graphs in Figure 4, we have used workspaces from size $9 \times 9$ till $50 \times 50$. In Figure 4(b) we observe the performance of $MT^*$ with the increase in workspace size for 2 and 3 robot systems for query $\phi_2$. 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 a linear increase in the computation time with the increase in the workspace size. These results are consistent for other queries and workspaces. In Figure 4(c) we observe that computation time of $MT^*$ increases exponentially with the increase in the number of robots for all the LTL queries. The graph shown is for query $\phi_2$. This is because, with the increase in the number of robots, the number of possible task assignments increases exponentially.

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 $|V|$ 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, 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.

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. As our experimental results establish, the computation time for $MT^*$ increases linearly with the increase in the size of the workspace. We were able to generate plans for up to 8 robots for $30 \times 30$ sized workspace, whereas the state-of-the-art algorithm hardly scales up to 3 robots over $15 \times 15$ sized workspace.

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 are done with 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 an anytime implementation (like anytime A* \[20\]). Moreover, it is possible to parallelize the evaluation of the suffix cycles (for different final states) in MT* to boost the performance further.

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 [46, 45, 19, 8, 31] will be employed to ensure collision avoidance among the robots.

In our future work, we plan to extend our algorithm to deal with dynamic obstacles \[26, 12\] and preferential constraints \[6, 3\]. We also plan to explore the possibility of applying the recently developed heuristics for multi-agent path finding [29] to make MT* more scalable.

REFERENCES

[1] F. Bacchus and F. Kabanza. Planning for temporally extended goals. *Ann. Math. Artif. Intell.*, 22(1-2):5–27, 1998.
[2] C. Baier and J.-P. Katoen. *Principles of Model Checking (Representation and Mind Series)*. The MIT Press, 2008.
[3] J. A. Baier, F. Bacchus, and S. A. McIlraith. A heuristic search approach to planning with temporally extended preferences. *Artif. Intell.*, 173(5-6):593–618, 2009.
[4] J. A. Baier and S. A. McIlraith. Planning with first-order temporally extended goals using heuristic search. In *AAAI*, pages 788–795, 2006.
[5] J. A. Baier and S. A. McIlraith. Planning with temporally extended goals using heuristic search. In *ICAPS*, pages 342–345, 2006.
[6] J. A. Baier and S. A. McIlraith. Planning with preferences. *AI Magazine*, 29(4):25–36, 2008.
[7] T. Balch and R. Arkin. Behavior-based formation control for multirobot teams. *IEEE Transaction on Robotics and Automation*, 14(6):926–939, 1998.
[8] A. Best, S. Narang, and D. Manocha. Real-time reciprocal collision avoidance with elliptical agents. In *ICRA*, pages 298–305, 2016.
[9] A. Bhatia, L. E. Kavraki, and M. Y. Vardi. Motion planning with hybrid dynamics and temporal goals. In *CDC*, pages 1108–1115, 2010.
[10] A. Bhatia, L. E. Kavraki, and M. Y. Vardi. Sampling-based motion planning with temporal goals. In *ICRA*, pages 2689–2696, 2010.
[11] A. Camacho, J. A. Baier, C. J. Muise, and S. A. McIlraith. Finite LTL synthesis as planning. In *ICAPS*, 2018.
[12] J. Cannon, K. Rose, and W. Ruml. Real-time motion planning with dynamic obstacles. In *SOCS*, 2012.
[13] Y. Chen, X. C. Ding, A. Stefanescu, and C. Belta. Formal approach to the deployment of distributed robotic teams. *IEEE Transactions on Robotics*, 28(1):158–171, 2012.
[14] Y. Chen, J. Tůmačová, and C. Belta. LTL robot motion control based on automata learning of environmental dynamics. In *ICRA*, pages 5177–5182, 2012.
[15] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms, Third Edition*. The MIT Press, 3rd edition, 2009.
[16] A. Duret-Lutz and D. Poitrenaud. Spot: An extensible model checking library using transition-based generalized buchi automata. In *MASCOTS*, 2004.
[17] D. Fox, W. Burgard, H. Kruppa, and S. Thrun. A probabilistic approach to collaborative multi-robot localization. *Autonomous Robots*, 8(3):325–344, 2000.
[18] 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*, pages 9–18, 1998.
[19] D. Hennes, D. Claes, W. Meeussen, and K. Tuyls. Multi-robot collision avoidance with localization uncertainty. In *AAMAS*, pages 147–154, 2012.
[20] J. S. Jennings, G. Whelan, and W. F. Evans. Cooperative search and rescue with a team of mobile robots. In *ICRA*, pages 193–200, 1997.
[21] Y. Kantaros and M. M. Zavlanos. Sampling-based optimal control synthesis for multirobot systems under global temporal tasks. *IEEE Trans. Autom. Control.*, 64(5):1916–1931, 2019.
[22] Y. Kantaros and M. M. Zavlanos. Stylus*: A temporal logic optimal control synthesis algorithm for large-scale multi-robot systems. *Int. J. Robotics Res.*, 39(7), 2020.
[23] S. Karaman and E. Frazzoli. Sampling-based motion planning with deterministic \(\mu\)-calculus specifications. In *CDC*, pages 2222–2229, 2009.
[24] D. Khalidi, D. Gujarathi, and I. Saha. T*: A heuristic search based algorithm for motion planning with temporal goals. In *ICRA*, 2020.
[25] M. Kloetzer and C. Belta. Automatic deployment of distributed teams of robots from temporal logic motion specifications. *IEEE Transactions on Robotics*, 26(1):48–61, 2010.
[26] S. Koenig and M. Likhachev. D*Lite. In *AAAI*, pages 476–483, 2002.
[27] H. Kress-Gazit, G. E. Fainekos, and G. J. Pappas. Where’s Waldo? Sensor-based temporal logic motion planning. In *ICRA*, pages 3116–3121, 2007.
[28] S. M. LaValle. *Planning Algorithms*. Cambridge University Press, New York, NY, USA, 2006.
[29] J. Li, A. Felnner, E. Boysarski, H. Ma, and S. Koenig. Improved heuristics for multi-agent path finding with conflict-based search. In *IJCAI*, pages 442–449, 2019.
[30] M. Likhachev, G. J. Gordon, and S. Thrun. ARA*: Anytime A* with provable bounds on sub-optimality. In *NIPS*, pages 767–774, 2003.
[31] P. Long, T. Fan, X. Liao, W. Liu, H. Zhang, and J. Pan. Towards optimally decentralized multi-robot collision avoidance via deep reinforcement learning. In *ICRA*,
