IEEE Robotics and Automation Letters (RA-L) paper, presented at ICRA 2026, Vienna, Austria. Cite as RA-L paper.

Online Modifications of High-Level Swarm Behaviors

Bo-Ruei Huang and Hadas Kress-Gazit, *Fellow, IEEE*

***Abstract*—Recent work has demonstrated how one can write high-level specifications for swarm behaviors and automatically create controllers for the individual robots to achieve the overall swarm task. In this letter, we address the question of how to modify, during execution, the desired behavior while maintaining guarantees on the behavior, if possible; we define three types of modification: changing the maximum number of robots in a region of the workspace, changing the connectivity of the workspace, and redistributing robots. During execution, if the specification is modified, we update the controller by creating local patches. Given the starting and ending state of the patch, we jointly use a symbolic synthesis tool and a constraint programming solver to synthesize robot control. We demonstrate our approach in simulation.**

***Index Terms*—Swarm robotics, formal methods in robotics and automation.**

I. Introduction

SWARM robotics focuses on the collective behavior of a large number of relatively simple robots [1]. While some approaches start with the control of individual robots in a swarm and analyze the emergent behavior [1], other approaches specify goals and/or constraints for the swarm as a whole and generate the individual robot control [2], [3], [4], [5], [6], [7]. One such top-down approach uses temporal logic to specify goals and constraints, and algorithms for model checking and synthesis to generate control. Kloetzer and Belta [5] presented a hierarchical framework for creating swarm controllers from Linear Temporal Logic (LTL). Haghighi et al. [8] encoded spatial-temporal logic specifications into a mixed integer programming (MIP) formulation to build a swarm robot planner. Yan et al. [9] proposed Swarm Signal Temporal Logic for monitoring swarm behaviors. Cardona et al. [10] controls swarms from Metric Temporal Logic, including splitting and merging subswarms. Leahy et al. [7] defined Capability Temporal Logic as specifications for the swarm coordination problem and provides a MIP formulation for control synthesis. Djeumou et al. [11] developed a probabilistic control algorithm for a swarm planning problem with high-level behaviors specified in Graph Temporal Logic. Moarref and Kress-Gazit [12] defined swarm behaviors using LTL and synthesized decentralized controllers that satisfy the specifications; Chen et al. [6] expanded the abstraction in [12] to include swarm formation control, and automatically designed continuous controllers for the swarm robots to fulfill the high-level task.

In the approaches above, the desired swarm behavior (specification) is given a priori and then synthesized and executed; the specification does not change during execution. In real deployments, there may be unexpected events, or the user may decide to change the specification on the fly. For example, there may be obstacles that block robots' routes, a room may be unexpectedly cluttered, preventing the originally assigned robots from fitting safely inside, or the user may prefer different numbers of robots in different rooms for a monitoring task. In this letter, we address the problem of updating the specification and control, if possible, in a provably correct way.

A swarm plan required to satisfy high-level specifications may be computationally expensive to synthesize, and a full re-synthesis may not be needed for a small modification in the specifications. Therefore, inspired by [13], we build local patches to adjust the current swarm plan. To deal with environmental events that caused the robot to violate a specification, the previous work in [13] provided a search algorithm that identifies a portion of the synthesized controller that needs re-synthesis, and proceeds with creating a patch that fixes the controller. In this letter, we allow the specification to be changed deliberately, not only to react to environmental events, and we create patches that do not depend only on physical distances; furthermore, they may include the swarm splitting and merging. Specifically, given the original swarm plan, our algorithm looks for sub-sequences of the plan that violate the new specification and uses a constraint programming (CP) solver to synthesize a patch, replacing the violating sub-sequences with satisfying ones, when possible. **Contribution**: In this work, we define three types of swarm specification modification and present an online patching method to change the swarm behavior to address the modified specification. While previous work focuses on online patching for single robot behaviors or swarm behavior synthesis from fixed, a priori specifications, our work enables users to change the swarm behavior during execution. We demonstrate these methods in simulation.

II. Preliminaries

We briefly introduce swarm specifications and the synthesis algorithm [12] for creating a high-level swarm plan.

A. Linear Temporal Logic and the GR(1) Fragment

We use a fragment of Linear Temporal Logic (LTL) to specify requirements on a swarm's behavior. The syntax of LTL is:

$$\varphi ::= \pi|\neg\varphi|\varphi \vee \varphi| \bigcirc \varphi|\varphi\mathcal{U}\varphi \quad (1)$$

Received 22 July 2025; accepted 1 December 2025. Date of publication 26 December 2025; date of current version 31 December 2025. This article was recommended for publication by Associate Editor C. Ioan Vasile and Editor L. Pallottino upon evaluation of the reviewers' comments. This work was supported by NSF under Grant IIS-1830471. *(Corresponding author: Bo-Ruei Huang.)*

The authors are with the Sibley School of Mechanical and Aerospace Engineering, Cornell University, Ithaca, NY 14853 USA (e-mail: bh574@cornell.edu; hadaskg@cornell.edu).

This article has supplementary downloadable material available at https://doi.org/10.1109/LRA.2025.3648501, provided by the authors.
Digital Object Identifier 10.1109/LRA.2025.3648501

2377-3766 © 2025 IEEE. All rights reserved, including rights for text and data mining, and training of artificial intelligence and similar technologies. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

©2026 IEEE

Authorized licensed use limited to: Cornell University Library. Downloaded on March 03,2026 at 20:11:45 UTC from IEEE Xplore. Restrictions apply.

IEEE Robotics and Automation Letters (RA-L) paper, presented at ICRA 2026, Vienna, Austria. Cite as RA-L paper.

Fig. 1. Example 1: (a) the region graph and capacities (b) swarm plan.

where $\pi \in \Pi$ is a Boolean proposition, $\neg$ is negation, $\vee$ is disjunction, $\bigcirc$ is next and $\mathcal{U}$ is until. From those, we define other operators such as conjunction $\wedge$, implication $\Rightarrow$, always $\square$, and eventually $\Diamond$.

The semantics of LTL are defined over the set of infinite words $\sigma = \sigma_0, \sigma_1, \sigma_2, \ldots, \sigma_i \in 2^{\Pi}$, where Π is the set of atomic propositions. The formula π is satisfied at step i if $\pi \in \sigma_i$, $\bigcirc\varphi$ is satisfied at step i if φ is satisfied in step $i+1$, $\square\varphi$ is satisfied if φ is satisfied in all steps and $\Diamond\phi$ is satisfied if φ is satisfied in the current or a future step. An infinite sequence satisfies φ, denoted as $\sigma \models \varphi$, if σ_0 satisfies it. We refer readers to [14] for more details.

In this letter, we consider LTL formulas of the form [15][1]:

$$\Phi = \theta_{init} \wedge \bigwedge_i \square\psi_i \bigwedge_j \square\Diamond\phi_j \tag{2}$$

where θ_{init} and ϕ_j are Boolean formulas over Π and ψ_i are Boolean formulas over $\Pi \cup \bigcirc\Pi$ where $\bigcirc\Pi$ are the propositions with the next operator. θ_{init} describes the initial conditions, ψ_i are safety guarantees which must always hold, and ϕ_j are liveness goals the system must repeatedly satisfy.

B. Swarm Specifications

We consider a swarm with a total of ρ robots moving in a partitioned workspace. The workspace is abstracted as a region graph $G_R = (R, E)$ that consists of a set of vertices $R = \{r_1 \ldots r_{|R|}\}$, representing $|R|$ regions in the workspace, and a set of edges E such that an edge $e_{i,j} = (r_i, r_j) \in E$ represents that robots can move between r_i and r_j without going through any other region. In addition, we define the set of capacities $C = \{c_1 \ldots c_{|R|}\}$ where c_i is the maximum number of robots that can be in region r_i simultaneously. To capture swarm behaviors in LTL, we define a set of atomic propositions $\pi_i \in \Pi$, which are true if and only if there is at least one robot inside region $r_i \in R$.

As in [6], [12], the swarm specification is of the form of (2) with initial conditions θ_{init}, safety guarantees ψ_i, and n livenesses $\{\phi_1, \cdots \phi_n\}$. To help filter out symbolic plans that are physically impossible to implement, we encode the connectivity of the workspace as safety guarantees restricting where robots can go. For each r_i:

$$\psi_i = \left(\bigcirc\pi_i \Rightarrow \bigvee_{e_{j,i} \in E} \pi_j\right) \wedge \left(\pi_i \Rightarrow \bigvee_{e_{i,j} \in E} \bigcirc\pi_j\right) \tag{3}$$

Example 1: Consider a workspace with five rooms, as shown in Fig. 1. We abstract the motion of the swarm with five propositions π_1, π_2, π_3, π_4, π_5 and build a region graph as shown in Fig. 1(a). Ellipses are labeled with the region name and its associated capacity, denoted as $r_i : c_i$.

Initially, there are five robots in r_1 and another five in r_2. The user has three requirements; robots must repeatedly all be in r_3, at least one robot must repeatedly go to r_5, and for safety reasons, when any robot is in r_5, at least one robot must stay at r_3. The specification is encoded as:

$$\Phi = (\pi_1 \wedge \pi_2 \wedge \neg\pi_3 \wedge \neg\pi_4 \wedge \neg\pi_5) \wedge \square(\pi_5 \Rightarrow \pi_3)$$
$$\wedge\square\Diamond(\neg\pi_1 \wedge \neg\pi_2 \wedge \pi_3 \wedge \neg\pi_4 \wedge \neg\pi_5) \wedge \square\Diamond(\pi_5) \tag{4}$$

In addition to the user-defined specifications in (4), we encode the region graph as safety guarantees. For example, for Region r_1, the formula is:

$$\psi_1 = (\bigcirc\pi_1 \Rightarrow (\pi_1 \vee \pi_2 \vee \pi_3 \vee \pi_4))\wedge$$
$$(\pi_1 \Rightarrow (\bigcirc\pi_1 \vee \bigcirc\pi_2 \vee \bigcirc\pi_3 \vee \bigcirc\pi_4)) \tag{5}$$

C. Swarm Behavior Synthesis

Given $|R|$ regions and time step t, we define state $p^t = (\rho_1^t \cdots, \rho_i^t, \ldots, \rho_{|R|}^t)$, where ρ_i^t is the number of robots assigned to region r_i at time t such that $\sum_i \rho_i^t = \rho$ for all t. The goal of the swarm behavior synthesis is to create an infinite sequence of states $\mathcal{P} = p^0 p^1 p^2 \cdots$, such that the swarm satisfies its specifications (Section II-B).

Given a state p^t, with a slight abuse of notation, we define $\sigma(p^t) = \{\pi_i \in \Pi | \rho_i^t > 0\}$ as the abstract state of the swarm, where $\sigma(p^t)$ is the set of propositions that are true at time step t, i.e. there is at least one robot in each region i such that $\pi_i \in \sigma(p^t)$.

Given $\mathcal{P}$, we define individual robot plans[2] $P_a = p_a^0 p_a^1 p_a^2 \cdots, a \in [1, \rho]$ where $\sigma(p_a^t)$ is a single proposition (corresponding to the location of robot a) and $\sigma(p^t) = \bigcup_{a \in [1,\rho]} \sigma(p_a^t)$.

Synthesis: Given an initial swarm state p^0, a user defined LTL specifications Φ (2), a region graph G_R, and region capacities C, synthesize individual robot plans $P_a, a \in [1, \rho]$ such that $\mathcal{P}$ satisfies the specifications: $\sigma = \sigma(p^0)\sigma(p^1)\ldots \models \Phi$ and $\forall i, t \geq 0$, $\rho_i^t \leq c_i$.

The algorithm in [6], [12] first creates a symbolic (abstract) plan through GR(1) synthesis [15], [16]; synthesis creates a strategy that includes symbolic transitions the swarm must take to complete the task; there might be multiple different paths in this strategy and the algorithm chooses one and then uses integer programming to assign robots to regions in each step to create P_a. Additionally, through the assignment process, we know how many robots move along an edge $(r_i, r_j) \in E$, in the transition from p^t to p^{t+1}, which we denote as $\rho_{i,j}^{t,t+1}$. In the following algorithms, we assume $\mathcal{P}$ also includes the transition information $\rho_{i,j}^{t,t+1}$. The synthesized swarm plan $\mathcal{P}$ has the following structure:

$$\mathcal{P} = \mathcal{P}_{prefix}\mathcal{P}_1 \cdots \mathcal{P}_n\mathcal{P}_1 \cdots \tag{6}$$

where $\mathcal{P}$ is a concatenation of sub-sequences of states $\mathcal{P}_k, k \in [1, n] \cup \{\text{prefix}\}$. The plan starts with a prefix sequence, $\mathcal{P}_{prefix}$, followed by suffix sequences $\mathcal{P}_1\mathcal{P}_2 \cdots \mathcal{P}_n$ that form a cycle. The last state in each $\mathcal{P}_k$ is a state that satisfies one of the liveness requirements ϕ_k.

[1]This formula is equivalent to the GR(1) formula [15] where the left side of the implication is $True$.

[2]We use a (agent) to denote individual robots, and r to denote regions.

Authorized licensed use limited to: Cornell University Library. Downloaded on March 03,2026 at 20:11:45 UTC from IEEE Xplore. Restrictions apply.

IEEE Robotics and Automation Letters (RA-L) paper, presented at ICRA 2026, Vienna, Austria. Cite as RA-L paper.

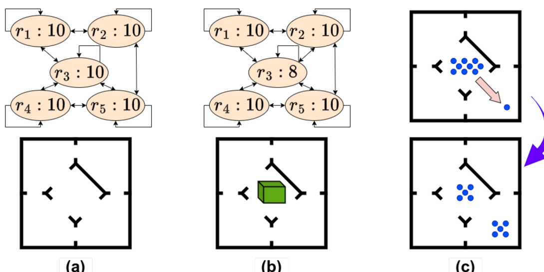

Fig. 2. Specification modifications for Example 1.

Example 1 (Continued): The synthesized symbolic swarm plan is $\sigma = \sigma_0\sigma_1\sigma_2\sigma_1\sigma_2\cdots$, where $\sigma_0 = \{\pi_1, \pi_2\}, \sigma_1 = \{\pi_3\}$, and $\sigma_2 = \{\pi_3, \pi_5\}$ and a feasible solution to the robot assignment is $\mathcal{P} = p^0p^1p^2p^1p^2\cdots$, or $\mathcal{P}_{prefix} = p^0$, $\mathcal{P}_1 = p^1$, and $\mathcal{P}_2 = p^2$, where $\mathcal{P}_1$, $\mathcal{P}_2$ forms a loop and p^3 returns to p^1. The assignment of robots becomes $p^0 = (5, 5, 0, 0, 0)$, $p^1 = (0, 0, 10, 0, 0)$, $p^2 = (0, 0, 9, 0, 1)$, which is illustrated in Fig. 1(b). The blue dots indicate the robot positions, and pink arrows indicate the directions the robots move in. Correspondingly, we create ten individual robot plans: one of them is $\pi_1\pi_3\pi_5$, four of them are $\pi_1\pi_3\pi_3$, and five of them are $\pi_2\pi_3\pi_3$. Therefore, $\rho^{0,1}_{1,3} = 5$, $\rho^{0,1}_{2,3} = 5$, $\rho^{1,2}_{3,3} = 9$, $\rho^{1,2}_{3,5} = 1$, $\rho^{2,3}_{3,3} = 1$ and $\rho^{2,3}_{5,3} = 1$.

III. Problem Formulation

In this work, we define three types of online specification modifications:

(i) **Region graph changes**: The user can change the set of edges $\hat{E}$ in the region graph by removing or adding edges. We assume R stays fixed.

$$\hat{G_R} = (R, \hat{E}) \tag{7}$$

(ii) **Region Capacity Changes**: The user can update the region capacities:

$$\hat{C} = \{\hat{c}_1 \dots \hat{c}_{|R|}\} \tag{8}$$

(iii) **Momentary Redistribution**: Given a state p^t, the user can momentarily redistribute the robots by requesting $\hat{p}^t$:

$$\hat{p}^t = (\hat{\rho}^t_1 \dots \hat{\rho}^t_{|R|}) \text{ s.t. } \sum_i \hat{\rho}^t_i = \rho \text{ and } \sigma(\hat{p}^t) = \sigma(p^t) \tag{9}$$

Example 1 (Continued): Example modification are shown in Fig. 2: (a) a door closing in the environment leads to a change in the region graph, $\hat{E} = E \backslash \{(r_2, r_3), (r_3, r_2)\}$, (b) a large obstacle reduces the capacity of room r_3 by 2, changing c_3 to $\hat{c}_3 = 8$ (c) during execution, the user decides that there should be 5 robots in room r_5, that is, for $p^2 = (0, 0, 9, 0, 1)$, the user requests $\hat{p}^2 = (0, 0, 5, 0, 5)$.

Problem statement: Given a nominal swarm plan $\mathcal{P}$ and P_a, the user defined specification Φ, and user defined modifications $\hat{G_R}$, $\hat{C}$, and/or $\hat{p}^t$ ((7) to (9)), synthesize an updated swarm behavior plan $\hat{\mathcal{P}}$ and the corresponding $\hat{P}_a$ that satisfy the modified specification, if feasible.

Synchronization assumption: In this work, we assume the robots are synchronized, i.e. they wait for each other to complete the current transition before continuing to the next. One can leverage synchronization skeletons [12] that relax this assumption and that causes robots to synchronize when needed and not at every step.

Algorithm 1: Swarm Plan Patching.

Input : $\hat{G_R}, \hat{C}, p^t, \hat{p}^t, \mathcal{P}, \Phi$
Output: $\{\hat{P}_a\}^{\rho}_{a=1}$
1 updateSuccess, $\hat{\mathcal{P}} \leftarrow$ UPDATE$(\hat{G_R}, \hat{C}, \mathcal{P}, \Phi)$
2 **if not** updateSuccess **then**
3 | **return failed, provide feedback to user**
4 **if** $\hat{p}^t = p^t$ **or** $p^t \notin \hat{\mathcal{P}}$ **then**
5 | $\{\hat{P}_a\}^{\rho}_{a=1} =$ SPLIT$(\hat{\mathcal{P}}, \hat{G_R})$
6 | **return** $\{\hat{P}_a\}^{\rho}_{a=1}$
/* Redistribution modifications */
7 solveSuccess, $\hat{\mathcal{P}} \leftarrow$ SOLVE_PATCH$(p^t, \hat{p}^t, \hat{G_R}, \hat{C}, \Phi, \hat{\mathcal{P}}, \top)$
8 **if** solveSuccess **then**
9 | updateSuccess, $\hat{\mathcal{P}} \leftarrow$ UPDATE$(\hat{G_R}, \hat{C}, \hat{\mathcal{P}}, \Phi)$
10 **if not** solveSuccess **or not** updateSuccess **then**
11 | **return failed, provide feedback to user**
12 $\{\hat{P}_a\}^{\rho}_{a=1} =$ SPLIT$(\hat{\mathcal{P}}, \hat{G_R})$
13 **return** $\{\hat{P}_a\}^{\rho}_{a=1}$

IV. Online Specification Modification

Our approach to solving the online swarm specification modification problem is presented in Alg. 1. Given a modification, we first check whether the current plan $\mathcal{P}$ is still valid; if parts of the plan no longer satisfy the (modified) specification, we create localized *patches* that repair the violations. We define a patch $\mathcal{P}_{p^s,p^f}$ as a sequence of states connecting two states p^s (start) and p^f (final).

In Alg. 1, we first create patches, if needed, based on changes in the region graph and/or the capacities (line 1). After the update, we check whether we need to redistribute the robots; if $\hat{p}^t$ is the same as p^t or p^t is not in the updated plan, we create and return the individual robot plans (lines 4-6). Otherwise, in line 7, we build a patch $\mathcal{P}_{p^t,\hat{p}^t}$ that redistributes the swarm from p^t to $\hat{p}^t$. After the redistribution, we may need to rebalance the number of robots assigned to each region in the plan. The rebalancing operation can lead to assignments that exceed the capacity specification, therefore we use $UPDATE(\hat{G_R}, \hat{C}, \mathcal{P}, \Phi)$ again to ensure all the region capacities are met. Finally, we use $SPLIT(\hat{\mathcal{P}}, \hat{G_R})$ to create separate plans for each robot, as discussed in Section V-D.

In $UPDATE(\hat{G_R}, \hat{C}, \mathcal{P}, \Phi)$ (Alg. 2), for each $\mathcal{P}_k \in \mathcal{P}$, we check whether any state or transition in $\mathcal{P}_k$ violates the region graph specifications; either exceeding region capacities or traversing edges that were removed (line 3). If there are any, we return $\tilde{\mathcal{P}}$, the set of all states $\{p^{t_1}, p^{t_2}, \cdots p^{t_v}\}$ that are in violation. Note that these states are not necessarily consecutive. After finding $\tilde{P}$, we set p^s, p^f to enclose all states between p^{t_1} and p^{t_v}. p^s is p^{t_1} if the swarm moves to the next state through a removed edge, or p^{t_1-1} if p^{t_1} violates region capacity constraints. Similarly, p^f is set to p^{t_v} or p^{t_v+1}. In

Authorized licensed use limited to: Cornell University Library. Downloaded on March 03,2026 at 20:11:45 UTC from IEEE Xplore. Restrictions apply.

IEEE Robotics and Automation Letters (RA-L) paper, presented at ICRA 2026, Vienna, Austria. Cite as RA-L paper.

Algorithm 2: $UPDATE(\hat{G_R}, \hat{C}, \mathcal{P}, \Phi)$.

Input : $\hat{G_R}, \hat{C}, \mathcal{P}, \Phi$
Output: `solveSuccess`, $\hat{\mathcal{P}}$

1. $\hat{\mathcal{P}} \leftarrow \mathcal{P}$
2. **for** $k \in \{prefix\} \cup [1, n]$ **do**
3. $\quad \tilde{\mathcal{P}} \leftarrow$ `CHECK_VIOLATION`$(\mathcal{P}_k, \hat{G_R}, \hat{C})$
4. $\quad$ **if** $\tilde{\mathcal{P}} \neq \emptyset$ **then**
5. $\quad\quad$ `expandSuccess`, $(p^s, p^f) \leftarrow$ `INITIALIZE_PATCH`$(\tilde{\mathcal{P}})$
6. $\quad\quad$ **while** `expandSuccess` **do**
7. $\quad\quad\quad$ `solveSuccess`, $\hat{\mathcal{P}} \leftarrow$ `SOLVE_PATCH`$(p^s, p^f, \hat{G_R}, \hat{C}, \Phi, \hat{\mathcal{P}}, \bot)$
8. $\quad\quad\quad$ **if** `solveSuccess` **then**
9. $\quad\quad\quad\quad$ **break**
10. $\quad\quad\quad$ **else**
11. $\quad\quad\quad\quad$ `expandSuccess`, $(p^s, p^f) \leftarrow$ `EXPAND`$(p^s, p^f, \hat{\mathcal{P}})$
12. $\quad\quad$ **if not** `expandSuccess` **then**
13. $\quad\quad\quad$ **return** `solveSuccess` $\leftarrow \bot, \hat{\mathcal{P}}$
14. **return** `solveSuccess` $\leftarrow \top, \hat{\mathcal{P}}$

line 7, $SOLVE_PATCH$ creates a new sequence to replace the original states and transition assignments between p^s, p^f, if it exists; if a patch is not found, we attempt to find a bigger patch using $EXPAND(p^s, p^f, \mathcal{P})$.

To limit computation time, when searching for patches we limit each patch to span two consecutive subsequences $\mathcal{P}_k$, $\mathcal{P}_{k+1}$. If p^s or p^f is not in $\mathcal{P}_k \cup \mathcal{P}_{k+1}$, $expandSuccess$ is False and the update stops. Each time no patch is found, we expand the size of the patch by setting $p^s \leftarrow p^{s-1}$ and $p^f \leftarrow p^{f+1}$, as long as both remain within $\mathcal{P}_k \cup \mathcal{P}_{k+1}$. If we cannot find a patch, we resynthesize the entire swarm behavior. While we restrict to two consecutive subsequences, it is straightforward to generalize the patch to include more subsequences; we omit the description for brevity.

V. Creating a Patch

Given the swarm specifications and patch boundaries (p^s, p^f), $SOLVE_PATCH(p^s, p^f, \hat{G_R}, \hat{C}, \Phi, \mathcal{P}, redist)$ synthesizes a state sequence connecting p^s to p^f (Alg. 3). If p^s and p^f belong to $\mathcal{P}_k$ and $\mathcal{P}_{k+1}$ respectively, the patch must also satisfy the liveness goal ϕ_k that is satisfied in $\mathcal{P}_k$; therefore we create patches that are composed of two parts, one reaching the liveness goal and the other reaching p^f [3]. Patch synthesis proceeds in two steps: first, we construct symbolic strategies $\Sigma(\phi), \Sigma(p^f)$ that ensure robot movements satisfy the specification (line 1, Section V-A); these are then encoded as constraints in the second step where we solve CP problems to assign robot counts to regions and determine their transitions (line 3, Section V-B).

Given $\Sigma(\phi)$ and $\Sigma(p^f)$, we determine the minimum (m_1, m_2) and maximum (M_1, M_2) lenght of paths that satisfy the specification. We use these to bound the patch size we are searching for. When $p^f \in \mathcal{P}_k$ instead of $\mathcal{P}_{k+1}$, we set $m_1 = M_1 = 0$ and $\phi = p^s$.

[3]To allow patches that span more subsequences, we can create a patch that includes multiple liveness goals and is composed of multiple parts.

Algorithm 3: $SOLVE_PATCH(p^s, p^f, \hat{G_R}, \hat{C}, \Phi, \hat{\mathcal{P}}, redist)$.

Input : $p^s, p^f, \hat{G_R}, \hat{C}, \Phi, \mathcal{P}$, `redist`
Output: `solveSuccess`, $\hat{\mathcal{P}}$

1. $m_1, M_1, \Sigma(\phi), m_2, M_2, \Sigma(p^f) \leftarrow$ `SYMBOLIC_PATCH`$(p^s, p^f, \hat{G_R}, \Phi)$
2. **while** $m_1 \leq M_1$ **and** $m_2 \leq M_2$ **do**
3. $\quad$ `synthesisSuccess`, $\hat{\mathcal{P}}_{p^s, p^f} \leftarrow$ `CP_SOLVE`$(p^s, p^f, \hat{G_R}, \hat{C}, m_1, m_2, \Sigma(\phi), \Sigma(p^f))$
4. $\quad$ **if** `synthesisSuccess` **then**
5. $\quad\quad$ **if** `redist` **then**
6. $\quad\quad\quad$ $p^{t_k} \leftarrow$ last state in $\mathcal{P}_k$ where p^s is in $\mathcal{P}_k$
7. $\quad\quad\quad$ $\bar{\mathcal{P}}_{prefix} \leftarrow \hat{\mathcal{P}}_{p^s, p^f}, p^{f+1}, \cdots, p^{t_k}$
8. $\quad\quad\quad$ $\bar{\mathcal{P}}_i \leftarrow \mathcal{P}_{i+k \pmod n}, \ \forall i \in [1, n]$
9. $\quad\quad\quad$ $\hat{\mathcal{P}} \leftarrow \bar{\mathcal{P}}_{prefix} \bar{\mathcal{P}}_1 \cdots \bar{\mathcal{P}}_n$
10. $\quad\quad\quad$ $\hat{\mathcal{P}} \leftarrow$ `REBALANCE`$(p^f, \hat{\mathcal{P}}, \hat{G_R}, \hat{C})$
11. $\quad\quad$ **else**
12. $\quad\quad\quad$ $\hat{\mathcal{P}} \leftarrow p^0, p^1, \cdots p^{s-1}, \hat{\mathcal{P}}_{p^s, p^f}, p^{f+1}, \cdots$
13. $\quad\quad$ **return** `solveSuccess` $\leftarrow \top, \hat{\mathcal{P}}$
14. $\quad$ **else**
15. $\quad\quad$ $m_1 \leftarrow$ **min**$(m_1 + 1, M_1)$
16. $\quad\quad$ $m_2 \leftarrow$ **min**$(m_2 + 1, M_2)$
17. **return** `solveSuccess` $\leftarrow \bot, \hat{\mathcal{P}}$

If we find a patch, we integrate it into the overall plan, returning $\hat{\mathcal{P}}$. In the case of redistribution, we reorder the swarm plan and rebalance the subswarm sizes (Alg. 3, line 6–10), as further explained in Section V-C.

A. Creating a Set of Abstract (symbolic) Patches

The first step in patch creation is computing symbolic strategies that constrain the CP to find solutions that satisfy the specification. To obtain $\Sigma(\phi)$ (or $\Sigma(p^f)$), we modify the GR(1) synthesizer [16] to create a startegy for a single goal (liveness) ϕ (or p^f), keeping the safety constraints from Φ, modified according to the region graph $\hat{G_R}$ (3). This strategy is encoded as a boolean formula over $\Pi \cup \bigcirc\Pi$ that represents all allowed transitions.

In addition, similar to [12], we too consider the safety of *intermediate states:* states that do not appear in the abstract plan, but do occur during execution. Consider a plan for which $\sigma(p^1) = \{\pi_i\}$ and $\sigma(p^2) = \{\pi_j\}$; this plan requires all robots to be in region i and then all robots to be in region j. A single robot can execute this plan, but a group of robots cannot, in practice, perform this transition. Instead, it will go through an intermediate state where some robots are in i and some are in j, i.e. $\sigma(p^{intermediate}) = \{\pi_i, \pi_j\}$.

In [12], the authors iteratively synthesized a symbolic swarm plan, checked its intermediate states against the safety specifications, and resynthesized with additional constraints if needed. Here, in contrast, we add to the GR(1) formula an additional safety specification that ensures the safety of the intermediate states.[4] This additional safety specification enumerates the allowable symbolic transitions $(\sigma(p^i), \sigma(p^{i+1})) \in safeTrans$

[4]In practice we add these constraints directly as a BDD when synthesizing the plan.

Authorized licensed use limited to: Cornell University Library. Downloaded on March 03,2026 at 20:11:45 UTC from IEEE Xplore. Restrictions apply.

IEEE Robotics and Automation Letters (RA-L) paper, presented at ICRA 2026, Vienna, Austria. Cite as RA-L paper.

that result in safe intermediate states, and is of the form:

$$\psi_{inter} = \Box \left(\bigvee_{safeTrans} \left(\bigwedge_{\pi_k \in \sigma(p^t)} \pi_k \bigwedge_{\pi_l \notin \sigma(p^t)} \neg \pi_l \bigwedge_{\pi_m \in \sigma(p^{t+1})} \bigcirc \pi_m \bigwedge_{\pi_n \notin \sigma(p^{t+1})} \neg \bigcirc \pi_n \right) \right) \quad (10)$$

For a given transition, there may be several possible intermediate states; we consider a transition safe if all of its intermediate states satisfy the specification. We define $e_{inter}(p^t, p^{t+1}) \subseteq \hat{E}$ as an intermediate transition, which is a collection of edges in $\hat{E}$ the swarm might traverse when transitioning between p^t and p^{t+1}. From example 1, we have $\sigma(p^0) = \{\pi_1, \pi_2\}$ and $\sigma(p^1) = \{\pi_3\}$, and correspondingly, $e_{inter}(p^0, p^1) = \{(r_1, r_3), (r_2, r_3)\}$.

Given $e_{inter}(p^t, p^{t+1})$, we follow the definition in [12] to formulate intermediate states as a boolean formula over Π:

$$\bigwedge_{(r_i, r_j) \in e_{inter}} (\pi_i \vee \pi_j) \bigwedge_{\{\pi_k \in \Pi \setminus (\sigma(p^t) \cup \sigma(p^{t+1}))\}} \neg \pi_k \quad (11)$$

Eq. (11) contains two parts: $\bigwedge_{(r_i, r_j) \in e_{inter}} (\pi_i \vee \pi_j)$ means each edge the robots travel through creates 3 possible states: r_i, $r_i \wedge r_j$, and r_j. $\bigwedge_{\pi_k \in \Pi \setminus (\sigma(p^t) \cup \sigma(p^{t+1}))} \neg \pi_k$ means the robots will not occupy regions that are unrelated to edges they traverse. Finally, for each possible transition $(\sigma(p^i), \sigma(p^{i+1}))$, we check whether (11) implies Φ; if so, we allow that transition by adding it to the set of safe transitions $safeTrans$. We then construct ψ_{inter} using (10) and add ψ_{inter} to our safety guarantees, used to create $\Sigma(\phi)$ ($\Sigma(p^f)$).

B. Synthesizing $\hat{\mathcal{P}}$

To synthesize a patch, we formulate the CP problem $CP_SOLVE(p^s, p^f, \hat{G_R}, \hat{C}, m_1, m_2, \Sigma(\phi), \Sigma(p^f))$ and solve it with miniZinc [17]. A valid patch $\hat{\mathcal{P}}_{p^s, p^f} = \hat{p}^0, \hat{p}^1, \hat{p}^2, \ldots$ starts at $p^0 = p^s$, visits ϕ, and ends at p^f. The CP solver assigns the number of robots in each region at every step and along each transition. The patch length is $m_1 + m_2 + 1$, and each consecutive state must satisfy $\Sigma(\phi)$ or $\Sigma(p^f)$ to respect safety constraints and ensure eventual visits to ϕ and p^f. We formalize the patch synthesis problem as follows:

Variables:

- **Region Assignments**: $\varrho_i^t \in [0, \hat{c}_i], t = [0, m_1 + m_2]$ captures how many robots should be in region r_i at the t-th state of the patch, so $\hat{p}^t = (\varrho_1^t, \varrho_2^t, \ldots)$.
- **Edge Assignments**: $\varrho_{i,j}^{t,t+1} \in [0, min(\hat{c}_i, \hat{c}_j)], t = [0, m_1 + m_2 - 1], (r_i, r_j) \in \hat{E}$ represents how many robots should move from region r_i to r_j between states t and $t+1$ in the patch.

Example 1 (modification a): The door between regions r_2, r_3 is closed. In this case, the transition between p^0 and p^1 violates the modified region graph–five robots move from r_2 to r_3–therefore we solve for a patch $\mathcal{P}_{p^0, p^1}$. In the following, we set $m_1 = 0, m_2 = 2$. Here the region variables are $\varrho_i^t \in [0, 10]$, for $i = [1, 5]$, $t = [0, 2]$, and the edge variable are $\varrho_{1,1}^{t,t+1} \in [0, 10], \varrho_{1,2}^{t,t+1} \in [0, 10]$, and so on.

Constraints:

- **Encoding $\Sigma(\phi)$ ($\Sigma(p^f)$):** For each step $t \in [0, m_1 - 1]$ ($[m_1, m_1 + m_2 - 1]$), consecutive states $\hat{p}^t$ and $\hat{p}^{t+1}$ must satisfy $\Sigma(\phi)$ ($\Sigma(p^f)$), a BDD-encoded boolean formula over $\Pi \cup \bigcirc\Pi$. Following [18], we translate $\Sigma(\phi)$ ($\Sigma(p^f)$) into boolean constraints over literals $\{\pi_i, \neg\pi_i, \bigcirc\pi_i, \neg\bigcirc\pi_i\}$, where each literal corresponds to $\{(\varrho_i^t > 0), (\varrho_i^t = 0), (\varrho_i^{t+1} > 0), (\varrho_i^{t+1} = 0)\}$, and auxiliary variables V_k representing the truth values of subformulae in $\Sigma(\phi)$ ($\Sigma(p^f)$).
- **Liveness Guarantee**: At $t = m_1$, the swarm reaches a state satisfying ϕ, represented as a boolean formula over Π. We convert this into equalities and inequalities as described above.
- **Patch Boundary**: $\varrho_i^0 = \rho_i^s$ and $\varrho_i^{m_1+m_2} = \rho_i^f$. The robot number assignment in the first and last state should be identical to p^s and p^f, respectively.
- **Robot Flow**: $\varrho_i^t = \sum_{(j,i) \in \hat{E}} \varrho_{j,i}^{t-1,t} = \sum_{(i,j) \in \hat{E}} \varrho_{i,j}^{t,t+1}$. The number of robots in a region should be the same as the number of robots entering or staying in that region during the previous transition and the number of robots exiting or staying in that region during the next transition.

In the returned modified plan $\hat{\mathcal{P}}$, the patch either replaces all states and transitions between p^s, p^f when solving for region graph updates (when $redist$ is false), or is attached as a prefix to $\hat{\mathcal{P}}$ when redistributing the robots.

Example 1 (modification a): The patching returns $\hat{\mathcal{P}}_{p^0, p^1}$ as $p^s = p^0 = (5, 5, 0, 0, 0), \hat{p}^{t_1} = (5, 0, 5, 0, 0), \hat{\rho}_{1,3}^{0,t_1} = 5, \hat{\rho}_{2,1}^{0,t_1} = 5$, $\hat{\rho}_{1,3}^{t_1,1} = 5$, and $p^f = p^1 = (0, 0, 10, 0, 0)$; that is, between $p^s, \hat{p}^{t_1}$, 5 robots go from r_1 to r_3, 5 robots go from r_2 to r_1, and between $\hat{p}^{t_1}, p^f$, 5 robots in r_1 go to r_3. The modified swarm plan than becomes $\hat{\mathcal{P}} = \hat{p}^0 \hat{p}^1 \hat{p}^2 \hat{p}^3 \hat{p}^2 \cdots$, where $\hat{p}^0 = p^0, \hat{p}^1 = \hat{p}^{t_1}, \hat{p}^2 = p^1, \hat{p}^3 = p^2$.

C. Redistribution

Rebistribution requests will typically result in the need to adjust the full swarm plan; the previous swarm states (number of robots in each region) will be modified, and robots will receive new individual plans. In Example 1 modification (c), $p^2 = (0, 0, 9, 0, 1)$ is modified to $\hat{p}^2 = (0, 0, 5, 0, 5)$. Originally, robots move from p^2 to p^3 adhering to the nominal assignments: $\rho_{3,3}^{2,3} = 9$ and $\rho_{5,3}^{2,3} = 1$, but that is inconsistent with $\hat{p}^2$. Therefore, when creating a patch for redistribution, we modify the nominal plan such that the redistribution patch from p^t to $\hat{p}^t$ becomes the beginning of the modified plan. The prefix of the modified plan after the redistribution goes from p^t to $\hat{p}^t$ and satisfies ϕ_k at p^{t_k} (line 7, Alg. 3). The swarm will continue to satisfy each liveness goal in the same order as before the modification, starting from ϕ_{k+1}. In line 10 in Alg. 3, we modify the robot assignment in $\hat{\mathcal{P}}$ such that it is consistent with $\hat{p}^t$ using $REBALANCE(\hat{p}^t, \hat{\mathcal{P}}, \hat{G_R}, \hat{C})$; the function takes $\hat{\mathcal{P}}$ and tries to update all states following $\hat{p}^t$ by formulating CP problems for each prefix/suffix $\hat{\mathcal{P}}_k$, similar to CP_SOLVE, and solving them sequentially. Since we want to avoid resynthesizing an entire swarm plan (which requires symbolic strategies that address all liveliness goals), $REBALANCE(\hat{p}^t, \hat{\mathcal{P}}, \hat{G_R}, \hat{C})$ will not change how robots

Authorized licensed use limited to: Cornell University Library. Downloaded on March 03,2026 at 20:11:45 UTC from IEEE Xplore. Restrictions apply.

IEEE Robotics and Automation Letters (RA-L) paper, presented at ICRA 2026, Vienna, Austria. Cite as RA-L paper.

move symbolically, and we temporarily ignore the region capacity specification $\hat{C}$. If the rebalanced plan exceeds $\hat{C}$, we will attempt to fix it using $UPDATE$ in Alg. 1, line 9. Suppose states in modified pre/suffix $\hat{\mathcal{P}}_k$ are $\hat{p}^\tau, \ldots, \hat{p}^{\tau+|\hat{\mathcal{P}}_k|-1}$, we formulate $REBALANCE(\hat{p}^t, \hat{\mathcal{P}}, \hat{G_R}, \hat{C})$ as follows:

Variables:

- **Region Assignments**: $\varrho_i^t \in [0, \rho], t = [0, |\hat{\mathcal{P}}_k| - 1]$. This is the same as CP_SOLVE, except that the region capacities constraints are removed.
- **Edge Assignments**: $\varrho_{i,j}^{t,t+1} \in [0, \rho], t = [0, |\hat{\mathcal{P}}_k| - 2]$ and for each i, j such that $(r_i, r_j) \in \hat{E}$ and $\hat{\rho}_{i,j}^{t,t+1} > 0$. $\hat{\rho}_{i,j}^{t,t+1}$ is the number of robots that travel through (r_i, r_j) between $\hat{p}^t$ and $\hat{p}^{t+1}$. Since we are not changing the symbolic part of the plan, edges that are not used by the swarm in $\hat{\mathcal{P}}_k$ will not be assigned with any robot in the rebalanced plan.

Constraints:

- **Patch Boundary**: For $\hat{\mathcal{P}}_{prefix}$, given $\hat{p}^t = (\hat{\rho}_1^t, \ldots, \hat{\rho}_{|R|}^t)$, we enforce $\varrho_i^0 = \hat{\rho}_i^t$. Otherwise, given $\hat{p}^\tau = (\hat{\rho}_1^\tau, \ldots, \hat{\rho}_{|R|}^\tau)$, $\varrho_i^0 = \hat{\rho}_i^\tau$. For $\hat{\mathcal{P}}_n$ (last suffix), we additionally enforce $\varrho_i^{|\hat{\mathcal{P}}_k|-1} = \hat{\rho}_i^{|\hat{\mathcal{P}}_{prefix}|}$ because the last state in the last suffix should be identical to the last state in the prefix to form a loop.
- **Robot Flow**: $\varrho_i^t = \sum_{(j,i)\in\hat{E}} \varrho_{j,i}^{t-1,t} = \sum_{(i,j)\in\hat{E}} \varrho_{i,j}^{t,t+1}$. This is the same as CP_SOLVE.

Finally, since we want to avoid robot assignments that exceed regional capacities, we define the optimization goal for the CP solver, shown in (12). As a note, since we rebalance first (by fixing the symbolic plan) and then patch the plan, we may miss possible swarm behavior modifications; in such cases, we will resynthesize the full behavior.

$$min\left(\sum_{t\in[0,|\hat{\mathcal{P}}_k|-2]}\ \sum_{(r_i,r_j)\in\hat{E}} max\left(0, \varrho_{i,j}^{t,t+1} - min\left(\hat{c}_i, \hat{c}_j\right)\right)\right) \tag{12}$$

D. Creating Individual Robot Plans $\hat{P}_a$ ($SPLIT(\hat{\mathcal{P}}, \hat{G_R})$)

Given an updated overall plan $\hat{\mathcal{P}}$, we decompose it into individual plans $\hat{P}_a$ for each robot a, allowing them to execute their tasks independently. Each individual plan $\hat{P}_a$ shares the same prefix length as the overall plan $\hat{\mathcal{P}}$, but its suffix may differ in length from that of another robot's plan $\hat{P}_{a'}$ or from the suffix of $\hat{\mathcal{P}}$ itself. This discrepancy arises because when $\hat{\mathcal{P}}$ is split among the robots, a robot may end its individual plan in a region different from the one where the shared prefix ends (the first state of the cyclic suffix). To handle this, robots exchange the suffix portions of their plans, ensuring that each robot starts the next cycle of the suffix in the region where it finished the previous one. This suffix exchange must be a permutation over the robots, so that each robot's suffix is reassigned to exactly one robot, preserving the collective execution of the overall plan $\hat{\mathcal{P}}$. If this exchange occurs periodically at the end of each suffix section, then the total length of an individual plan $\hat{P}_a$ becomes $|\hat{\mathcal{P}}| + I \times (\sum_{k=1}^n |\hat{\mathcal{P}}_k|)$ where I is a non-negative integer denoting the number of exchange rounds.

In Alg. 4 line 1, we first initialize $\hat{P}_a$ for each robot a with the region it is in in the initial state $\hat{p}^0$. In lines 2-7, we create $\hat{P}_a$ for each robot a by distributing the transitions in $\hat{\mathcal{P}}$. For each transition between $\hat{p}^t$ and $\hat{p}^{t+1}$ in $\hat{\mathcal{P}}$, and for each individual robot plan, given $\hat{p}_a^t = \{\pi_i\}$, if we have robot transferring from region r_i to r_j, we append $\hat{p}_a^{t+1} = \{\pi_j\}$ to the end of $\hat{P}_a$. In lines 8-13, for each individual robot plan, we check if the suffix ends in the same region as where the suffix starts. If not, we determine which suffix each robot must append ($addSuffix$) by looking for $\hat{P}_{a'}$ whose suffix starts in the same region as the end region of $\hat{P}_a$ (lines 14-18). Once we obtain the suffix ordering, we unroll $addSuffix$ to finish building the complete $\hat{P}_a$, lines 19-24.

In Example 1, for all the individual plans, the last region in the plan is also the first in the cycle, so no suffix exchange is neccessary. To illustrate suffix exchange, we consider the following nominal plan for the same region map, initial condition, and swarm size as Example 1. The new nominal plan is $\mathcal{P} = p^0p^1p^2p^3p^1\cdots$, where $p^0 = (5,5,0,0,0)$, $p^1 = (0,0,9,0,1)$, $p^2 = (0,9,0,1,0)$, and $p^3 = (5,5,0,0,0)$. Here the prefix $\mathcal{P}_{prefix} = p^0$ has length 1, and the overall plan length is $|\mathcal{P}| = 4$. We split $\mathcal{P}$ into 10 individual plans.

In Alg. 4, lines 2-7 create the individual plans: $P_1 = \pi_1\pi_3\pi_4\pi_1$, $P_2 = P_3 = P_4 = P_5 = \pi_1\pi_3\pi_2\pi_2$, $P_6 = \pi_2\pi_5\pi_2\pi_2$, $P_7 = P_8 = P_9 = P_{10} = \pi_2\pi_3\pi_2\pi_1$. As can be seen, P_1 starts and ends in π_1, therefore, $addSuffix[1] = 1$ and there is no need to add an additional suffix to Robot 1's plan. Same goes for Robot 6. Robot 2's plan is $P_2 = \pi_1\pi_3\pi_2\pi_2$; this path does not return to its starting region π_1. Meanwhile Robot 7's plan is $P_7 = \pi_2\pi_3\pi_2\pi_1$. Following lines 8-18, we set $addSuffix[2] = 7$, adding to Robot 2's plan Robot 7's suffix, and so on. The resulting suffix permutation table for this example is $addSuffix = \{1,7,8,9,10,6,2,3,4,5\}$ and the final plans for the robots are $P_1 = \pi_1\pi_3\pi_4\pi_1$, $P_2 = P_3 = P_4 = P_5 = \pi_1\pi_3\pi_2\pi_2\pi_3\pi_2\pi_1$, $P_6 = \pi_2\pi_5\pi_2\pi_2$, $P_7 = P_8 = P_9 = P_{10} = \pi_2\pi_3\pi_2\pi_1\pi_3\pi_2\pi_2$.

VI. Theoretical Analysis

Soundness and completeness of Patching: The patching mechanism is sound as long as the patch covers all states and transitions that violate the specification following the modification. Given p^s, ϕ, p^f and unbounded M_1, M_2, if a patch exists, we will find it since $\Sigma(\phi)$ and $\Sigma(p^f)$ include, for every symbolic swarm state, all possible next symbolic states reachable to ϕ and p^f. Similarly, if we allow patches to span multiple subsequences, if a patch exists, we will find it. However, the patching approach is not complete, even if we allow patches of any size, because we restrict patches to be sequences and not cycles; if the entire plan suffix needs to be replaced, there is no p^f and our approach will fail.

Complexity of Patching: If we restrict each patch to two consecutive segments, $\mathcal{P}_k, \mathcal{P}_{k+1}$, the maximum number of patch expansions from an initial patch in $\mathcal{P}_k$ is $\frac{1}{2}(|\mathcal{P}_k| + |\mathcal{P}_{k+1}|)$. Assuming the longest symbolic sequence to any swarm state is M, the CP solver is called at most M times per patch. Each CP problem contains at most $2M$ transitions, requiring $|R|(2M+1)$ region variables, $2|\hat{E}|M$ edge variables, and $2|V_k|M$ boolean auxiliary variables per step. It gives a worst-case complexity of $O(\rho^{(|R|(2M+1)+2|\hat{E}|M)}2^{2|V_k|M})$ since constraint satisfaction is NP-complete [19]. Thus, the overall worst-case complexity is $O(\sum_k \frac{1}{2}(|\mathcal{P}_k| + |\mathcal{P}_{k+1}|)M\rho^{(|R|(2M+1)+2|\hat{E}|M)}2^{2|V_k|M})$.

Authorized licensed use limited to: Cornell University Library. Downloaded on March 03,2026 at 20:11:45 UTC from IEEE Xplore. Restrictions apply.

IEEE Robotics and Automation Letters (RA-L) paper, presented at ICRA 2026, Vienna, Austria. Cite as RA-L paper.

Algorithm 4: $SPLIT(\hat{\mathcal{P}}, \hat{G_R})$.

Input : $\hat{\mathcal{P}}, \hat{G_R}$
Output: $\{\hat{P}_a\}_{a=1}^{\rho}$

1 $\{\hat{P}_a\}_{a=1}^{\rho} \leftarrow$ `INITIALIZE`$(\hat{p}^0)$
2 **for** $t = [0..|\hat{\mathcal{P}}| - 1]$ **do**
3 $\quad \tilde{\rho}_{i,j} \leftarrow \rho_{i,j}^{t,t+1}$ for each $\hat{e}_{i,j} \in \hat{E}$
4 $\quad$ **for** $a = [1..\rho]$ **do**
$\quad\quad$ `/* suppose` $\hat{P}_a$ `ends in` π_i `*/`
5 $\quad\quad$ find $\hat{e}_{i,j}$ such that $\tilde{\rho}_{i,j}$ is not zero
6 $\quad\quad \hat{P}_a \leftarrow \hat{P}_a \pi_j$
7 $\quad\quad \tilde{\rho}_{i,j} \leftarrow \tilde{\rho}_{i,j} - 1$
8 `addSuffix` $\leftarrow$empty array of length ρ
9 `checked` $\leftarrow \{\}$
10 **for** $a = [1..\rho]$ **do**
11 $\quad$ **if** $\hat{p}_a^{|\hat{P}_a|-1} = \hat{p}_a^{|\hat{\mathcal{P}}_{prefix}|-1}$ **then**
12 $\quad\quad$ `checked` $\leftarrow$ `checked` $\cup \{a\}$
13 $\quad\quad$ `addSuffix`$[a] \leftarrow a$
14 **for** $a = [1..\rho]$ **do**
15 $\quad$ **for** $a' = [1..\rho]$ **do**
16 $\quad\quad$ **if** $\hat{p}_a^{|\hat{P}_a|-1} = \hat{p}_{a'}^{|\hat{\mathcal{P}}_{prefix}|-1}$ **and** $a' \notin$ `checked` **and** `addSuffix`$[a]$ is empty **then**
17 $\quad\quad\quad$ `checked` $\leftarrow$ `checked` $\cup \{a'\}$
18 $\quad\quad\quad$ `addSuffix`$[a] \leftarrow a'$
19 **for** $a = [1..\rho]$ **do**
20 $\quad \alpha \leftarrow$ `addSuffix`$[a]$
21 $\quad$ **while** $\alpha \neq a$ **do**
22 $\quad\quad \hat{P}_\alpha^{suffix} = \hat{p}_\alpha^{|\hat{\mathcal{P}}_{prefix}|} \dots \hat{p}_\alpha^{|\hat{\mathcal{P}}|-1}$
23 $\quad\quad \hat{P}_a \leftarrow \hat{P}_a \hat{P}_\alpha^{suffix}$
24 $\quad\quad \alpha \leftarrow$ `addSuffix`$[\alpha]$
25 **return** $\{\hat{P}_a\}_{a=1}^{\rho}$

Soundness and completeness of $SPLIT(\hat{\mathcal{P}}, \hat{G_R})$**:** Given a region graph $\hat{G_R}$ and a swarm plan $\hat{\mathcal{P}}$ forming a "lasso" with $\varrho_i^{|\hat{\mathcal{P}}|-1} = \varrho_i^{|\hat{\mathcal{P}}_{prefix}|-1}$, splitting $\hat{\mathcal{P}}$ into ρ individual plans using Alg. 4 is sound and complete. Lines 2–7 ensure each individual plan has length $|\hat{\mathcal{P}}|$. Since robots may occupy different regions at $t = |\hat{\mathcal{P}}_{prefix}| - 1$ and $t = |\hat{\mathcal{P}}| - 1$, $addSuffix$ maps robots ending in r_i at $|\hat{\mathcal{P}}| - 1$ to those starting in r_i at $t = |\hat{\mathcal{P}}_{prefix}| - 1$. $\varrho_i^{|\hat{\mathcal{P}}|-1} = \varrho_i^{|\hat{\mathcal{P}}_{prefix}|-1}$ ensures this mapping to be a permutation of size ρ, and concatenating suffixes by $addSuffix$ yields finite $\hat{P}_a$ since all cycles in a finite permutation are finite. Finally, $\hat{p}_a^{|\hat{\mathcal{P}}|-1} = \hat{p}_a^{|\hat{\mathcal{P}}_{prefix}|-1}$ because the last suffix appended in the loop (lines 19–24) ends in the same region as $\hat{p}_a^{|\hat{\mathcal{P}}|-1}$.

Complexity of $SPLIT(\hat{\mathcal{P}}, \hat{G_R})$**:** The time complexity of $SPLIT(\hat{\mathcal{P}}, \hat{G_R})$ (Alg. 4) is $O(\rho^2 + \rho|\hat{\mathcal{P}}|)$. Lines 2–7 require $\rho \times |\hat{\mathcal{P}}|$ iterations to divide $\hat{\mathcal{P}}$ into individual plans $\hat{P}_a$. Lines 14–18 take ρ^2 iterations to construct $addSuffix$. Finally, lines 19–24 iterate over a permutation table of size ρ for ρ robots, the total iteration time is also up to ρ^2 times.

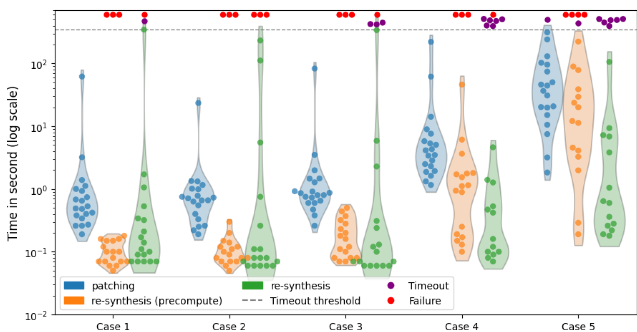

Fig. 3. Time spent on patching or re-synthesizing following removal of an edge in the region graph. The x-axis shows cases of increasing difficulty. Blue points represent our patching method, orange points re-synthesis with precomputed ψ_{inter}, and green points re-synthesis using [12]. At the top, red dots indicate failures (algorithm terminated without a solution), and purple dots mark runs exceeding 350 seconds. While re-synthesis is faster when successful, it has higher failure and timeout rates in all cases.

VII. Comparison to Re-Synthesis

We compare our patching approach to full re-synthesis of the swarm behavior [12].

Region capacity changes and redistribution: The approach in [12] first finds a symbolic plan and then solves an integer program to determine subswarn sizes; if the integer program fails, there is no mechanism to encode additional constraints into the symbolic synthesis. Therefore, [12] can only address region capacity changes that do not require changes in the symbolic plan, and cannot generally address redistribution. In contrast, our approach solves for the subswarm sizes while taking into account all possible symbolic plans, and can naturally address such modifications.

Region graph changes: To evaluate how specification complexity affects patching efficiency, we conduct an empirical study of a 50 robots swarm ($\rho = 50$) using randomly generated LTL specifications with varying map sizes $|R|$, number of safety requirements l, and number of liveness guarantees n. We use grid maps $R = \{r_{i,j} \mid i \in [1, H], j \in [1, W]\}$ of sizes 4×3, 4×4, and 5×4, where robots move between adjacent cells. Safety guarantees are either conjunctive $\psi_{conj} = \neg(r_{i_1,j_1} \wedge r_{i_2,j_2})$ or implicative $\psi_{impl} = r_{i_1,j_1} \Rightarrow r_{i_2,j_2}$, with r_{i_1,j_1} and r_{i_2,j_2} chosen randomly. Each liveness formula includes three random regions $R_k^s \subset R$ and $\phi_k = \bigwedge_{r \in R_k^s} r \bigwedge_{r \in R \setminus R_k^s} \neg r$. We test five cases: (1) $|R| = 12, l_{conj} = 3, n = 3$; (2) $|R| = 12, l_{conj} = 2, l_{impl} = 1, n = 3$; (3) $|R| = 12, l_{impl} = 3, n = 3$; (4) $|R| = 16, l_{impl} = 4, n = 4$; and (5) $|R| = 20, l_{impl} = 4, n = 5$. We omit specifications that are unrealizable.

Given the specifications and a valid nominal swarm plan, we compare the proposed patching method with resynthesis [12] and resynthesis using precomputed ψ_{inter} (Section V-A), for graph modifications where we remove one edge in the graph. For each case, we test 20 specifications and nominal plans, and record the computation time for all three methods. Fig. 3 shows the results: violin plots (blue—proposed patching, orange—resynthesis with ψ_{inter}, green—resynthesis from [12]) display the time distribution of successful runs, while purple and red dots indicate time-outs and failures, respectively.

Fig. 3 shows that while re-synthesis can be faster in successful cases, it fails in cases where patching succeeds, and it times out more often as the specification complexity increases (cases

Authorized licensed use limited to: Cornell University Library. Downloaded on March 03,2026 at 20:11:45 UTC from IEEE Xplore. Restrictions apply.

IEEE Robotics and Automation Letters (RA-L) paper, presented at ICRA 2026, Vienna, Austria. Cite as RA-L paper.

3–5). Even with precomputed ψ_{inter}, failures remain because some symbolic plans admit no valid robot assignments. The results also show that for patching, patch expansion can be time-consuming: in cases 1–4, a few outliers take 10–100× longer due to repeated CP solver time-outs and patchs that require expansion.

VIII. Demonstration

We demonstrate our approach in simulation, as seen in the accompanying video. The swarm of 50 robots is moving in an environment with 11 regions, and the nominal specification includes two safety and four liveness requirements. All computation is done on a personal computer equipped with an Intel Core i7 1370P processor.

The first modification includes capacities and connectivity changes in the region graph. Patching took 53.2 seconds to build a centralized plan $\hat{\mathcal{P}}$ and 0.03 seconds to split the plan. The second modification showcases redistribution, and took 72.9 seconds to build the centralized plan $\hat{\mathcal{P}}$ and 0.03 seconds to split the plan.

IX. Conclusion

We present a framework for autonomous online modification of robot swarm behavior, addressing three distinct types of modifications: changes in region capacities, region graph connectivity, and robot assignments. We create a patching mechanism to identify and rectify any swarm state within the plan that violates the modified specifications. We resolve these violations by generating new sequences of states using symbolic synthesis and a Constraint Programming solver. To showcase the effectiveness of our approach, we provide a demonstration that covers all three types of modifications. This demonstration illustrates the framework's ability to adapt high-level swarm behaviors in response to changes in specifications. The problem formulation can potentially be extended to full LTL, although doing so would require an efficient representation of symbolic strategies similar to $\Sigma(\phi)$. Other future work includes enabling users to specify minimal robot numbers, not only maximum capacities, solving for multiple liveness guarantees inside a patch, and reacting to environmental inputs or robot failures during execution.

References

[1] M. Brambilla, E. Ferrante, M. Birattari, and M. Dorigo, "Swarm Robotics: A review from the swarm engineering perspective," *Swarm Intell.*, vol. 7, pp. 1–41, Mar. 2013.

[2] B. Araki, J. Strang, S. Pohorecky, C. Qiu, T. Naegeli, and D. Rus, "Multi-robot path planning for a swarm of robots that can both fly and drive," in *Proc. 2017 IEEE Int. Conf. Robot. Automat.*, 2017, pp. 5575–5582.

[3] Y. Zhang and L. E. Parker, "Multi-robot task scheduling," in *Proc. 2013 IEEE Int. Conf. Robot. Automat.*, 2013, pp. 2992–2998.

[4] R. Luna and K. E. Bekris, "Efficient and complete centralized multi-robot path planning," in *Proc. 2011 IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2011, pp. 3268–3275.

[5] M. Kloetzer and C. Belta, "Temporal logic planning and control of robotic swarms by hierarchical abstractions," *IEEE Trans. Robot.*, vol. 23, no. 2, pp. 320–330, Apr. 2007.

[6] J. Chen, R. Sun, and H. Kress-Gazit, "Distributed control of robotic swarms from reactive high-level specifications," in *Proc. IEEE 17th Int. Conf. Automat. Sci. Eng.*, Lyon, France, Aug. 2021, pp. 1247–1254.

[7] K. Leahy et al., "Scalable and robust algorithms for task-based coordination from high-level specifications (ScRATCHeS)," *IEEE Trans. Robot.*, vol. 38, no. 4, pp. 2516–2535, Aug. 2022.

[8] I. Haghighi, S. Sadraddini, and C. Belta, "Robotic swarm control from spatio-temporal specifications," in *Proc. IEEE 55th Conf. Decis. Control*, Las Vegas, NV, USA, Dec. 2016, pp. 5708–5713.

[9] R. Yan, Z. Xu, and A. Julius, "Swarm signal temporal logic inference for swarm behavior analysis," *IEEE Robot. Autom. Lett.*, vol. 4, no. 3, pp. 3021–3028, Jul. 2019.

[10] G. A. Cardona, K. Leahy, and C.-I. Vasile, "Temporal logic swarm control with splitting and merging," in *Proc. 2023 IEEE Int. Conf. Robot. Automat.*, 2023, pp. 12423–12429.

[11] F. Djeumou, Z. Xu, M. Cubuktepe, and U. Topcu, "Probabilistic control of heterogeneous swarms subject to graph temporal logic specifications: A decentralized and scalable approach," *IEEE Trans. Autom. Control*, vol. 68, no. 4, pp. 2245–2260, Apr. 2023.

[12] S. Moarref and H. Kress-Gazit, "Automated synthesis of decentralized controllers for robot swarms from high-level temporal logic specifications," *Auton. Robots*, vol. 44, pp. 585–600, Mar. 2020.

[13] S. C. Livingston, R. M. Murray, and J. W. Burdick, "Backtracking temporal logic synthesis for uncertain environments," in *Proc. 2012 IEEE Int. Conf. Robot. Automat.*, St Paul, MN, USA, May 2012, pp. 5163–5170.

[14] E. M. Clarke, E. A. Emerson, and J. Sifakis, "Model checking: Algorithmic verification and debugging," *Commun. ACM*, vol. 52, pp. 74–84, Nov. 2009.

[15] R. Bloem, B. Jobstmann, N. Piterman, A. Pnueli, and Y. Sa'ar, "Synthesis of reactive(1) designs," *J. Comput. System Sci.*, vol. 78, pp. 911–938, May 2012.

[16] R. Ehlers and V. Raman, "Slugs: Extensible GR(1) synthesis," in *Proc. Comput. Aided Verification, in Lecture Notes in Computer Science*, S. Chaudhuri and A. Farzan, Eds., Berlin, Germany: Springer, 2016, pp. 333–339.

[17] N. Nethercote, P. J. Stuckey, R. Becket, S. Brand, G. J. Duck, and G. Tack, "MiniZinc: Towards a standard CP modelling language," in *Proc. Princ. Pract. Constraint Program.–CP, in Lecture Notes in Computer Science*, C. Bessière, Ed., Berlin, Germany: Springer, 2007, pp. 529–543.

[18] N. Eén and N. Sörensson, "Translating Pseudo-Boolean constraints into SAT," *J. Satisfiability, Boolean Model. Comput.*, vol. 2, pp. 1–26, Mar. 2006.

[19] A. A. Bulatov, "Constraint satisfaction problems: Complexity and algorithms," *ACM SIGLOG News*, vol. 5, pp. 4–24, Nov. 2018.

Authorized licensed use limited to: Cornell University Library. Downloaded on March 03,2026 at 20:11:45 UTC from IEEE Xplore. Restrictions apply.