Pick-Up and Delivery Planning in Multi-Agent Systems under Temporal Logic Specifications

ETIENNE DARGAUD

Examensarbete
Stockholm, Sverige December 2013

XR-EE-RT 2013:030
Abstract

This thesis describes an approach for solving planning problems for a team of robots involving picking up and moving objects. The temporal goals are described using linear temporal logic over subsets of the workspace. Temporal logic is a convenient formalism to capture the usual control specifications such as reachability and invariance as well as more complex specifications like sequencing and obstacle avoidance. Those goals include the positions of the robots and the positions of the objects, e.g., go to a given location, bring an object there or pick up this object. We consider robots moving in a 2D environment which is partitioned into non-overlapping regions. A cost representing the total number of actions done by all the robots will be used to evaluate the quality of the runs.

Our goal is to create a plan in a decentralized way for the robots that has a small cost and makes the robots collaborate to decrease the cost even more.

The approach is divided in several steps. First a plan is created for each robot using an abstraction of the robot and its specification. This plan describes the sequence of abstract actions the robot will do to fulfil its specifications. Second those actions are ordered in order to make their global cost as small as possible. Third and finally, the robot path for executing each action is found using automata theory. Such path is guaranteed to respect the global specification of the robot.

We support the proposed theory with experimental results, showing that it gives an optimal solution in some simple examples.
Contents

1 Introduction 3
   1.1 Related previous work 3
   1.2 Motivation 3
   1.3 Contribution 3

2 Preliminaries 5
   2.1 Linear temporal logic 5
   2.2 Transition system 5
   2.3 Büchi automaton 6
   2.4 Model-checking 6

3 Problem Formulation 8
   3.1 Problem set-up 8
   3.2 Robot model 8
   3.3 Specifications 9
   3.4 Run of the team of robots 9
   3.5 Cost of the run of the team of robots 10
   3.6 Problem statement 11
   3.7 Examples 11

4 Solution 15
   4.1 Plan’s creation 15
   4.2 Synchronization 17
   4.3 Ordering actions 19

5 Execution 24
   5.1 Cooperation 26

6 Illustration 27

7 Discussion 32

8 Conclusions and Future Work 33
1 Introduction

1.1 Related previous work

Robot motion planning is a well-explored area. It is traditionally A to B planning. A robot has to find a path for satisfying its specifications that are composed of safety conditions such as obstacle avoidance, reachability and/or surveillance: a robot has to go to a given region infinitely many times.

Temporal-logic-based motion planning provides a correct by design controller synthesis for autonomous robots. Temporal logics such as Linear Temporal Logic (LTL) and Computational Tree Logic (CTL) provide high level languages that can describe complex objectives. There are a lot of methods for dealing with motion planning under LTL specifications for a single robot [2], [3], [6] and for a team of robots [4]. Most of them use model checking [1] to find a controller but others model the problem as a two players game and look for a winning strategy [9]. When considering a team of robots, a lot of research focus on centralized solutions [8], [4] while other prefer decentralized solutions [7].

All those methods use a hierarchical approach which consists of three steps. The first step consists of partitioning the workspace in non-overlapping regions. The existence of a controller that can bring a robot from one region to an adjacent one is assumed. The second step is to create a discrete plan for the robot that consists of a sequence of regions to visit. The last step is to execute this plan using a hybrid controller and low level controllers that can bring the robot from one region to an adjacent region. In this work, we are only interested in the discrete planning and we will assume that the system is already discretized.

1.2 Motivation

Moving object is a task that a lot of robots have to do, to bring objects to elderly or disabled people or to move objects in a warehouse for example. We would like to develop a solution to make a team of robots move objects while respecting their specifications. Those specifications can be for example to move objects to some regions in a given order or make an object stay in a given region until another object has reached its destination.

We propose a decentralized solution which allows for more flexibility: robots can leave or join the team at any time without having to recompute a solution from scratch. To our best knowledge, such a problem has not been addressed in literature before.

1.3 Contribution

Here we describe a decentralized solution for a team of robot where each robot has its own specification which may involve moving objects. The abstract mobility capability of each robot is modelled by a transition system. The states of the transition system correspond to the different positions possible for the robot and the objects. Each transition is given a cost. The specifications are given in the form of co-safe $LTL_{\neg x}$ formulas. Our goal is to minimize the sum of the cost of the actions of all the robots while satisfying the specifications. We will not find the solution with the smallest cost but we try to get as close as
possible.
The first step is to create a transition system which is not constrained by the motion capability of the robot. In this transition system, we consider only one robot and a transition corresponds to the robot moving at most one object but it can move it to any region, adjacent or not. From this transition system, we will extract a plan of the form of a sequence of states which satisfies the specification of the robot. Then locks will be added to this plan to prevent other robots from moving an object and violating the specification. The actions of the robots are then ordered to try to reduce their global cost. Finally, before doing an action, a robot will ask the other robots if they can help him and reduce the cost of the action.
2 Preliminaries

In this chapter, we introduce necessary notation. We provide a formal definition of linear temporal logic, transition system and model-checking to describe the properties we are interested in. The definitions are taken from [1] and [12].

2.1 Linear temporal logic

Linear Temporal Logic (LTL) is an extension of propositional logic suitable for reasoning about infinite sequences of states. LTL formulas are defined over a set of Atomic Propositions $AP$. Here we will only use a subset of the LTL formulas. We will restrict our attention to a special kind of LTL formulas called co-safe $LTL_{-X}$ which are suitable for finite sequences of states. They are built using propositional operators ($\neg$, $\land$, $\lor$, i.e, negation, conjunction, disjunction) and temporal operators ($U$, $\diamond$, i.e until, eventually). In order to use co-safe formulas, the negation operator can only be used in front of atomic propositions.

LTL formulas are interpreted over sequences $\sigma: N \rightarrow 2^{AP}$. Let $\Phi_{AP}$ be the set of all those sequences. LTL formulas are evaluated by starting its interpretation from $\sigma(0)$.

Formula $\phi_1 U \phi_2$ holds if $\phi_1$ holds until $\phi_2$ becomes true which is required to happen. Formula $\diamond \phi$ holds if $\phi$ becomes true in some future time.

Definition 1. Co-safe $LTL_{-X}$ syntax

$$\phi := p | \neg p | \phi_1 \lor \phi_2 | \phi_1 \land \phi_2 | \phi_1 U \phi_2$$

where $p \in AP$.

Given a sequence $\sigma = \sigma(0), \sigma(1), \ldots, \sigma(n)$ and a LTL formula $\phi$ we denote $\sigma \models \phi$ when the evaluation of $\phi$ over $\sigma$ is true. In the following, $p \in AP$ and $\phi_1, \phi_2 \in \Phi_{AP}$ and for $i, k \in N, \sigma^i$ is defined as $\sigma^i(k) = \sigma(k+i)$.

$\sigma \models true$

$\sigma \models false$

$\sigma \models p$ iff $p \in \sigma(0)$

$\sigma \models \neg p$ iff $p \not\in \sigma(0)$

$\sigma \models \phi_1 \land \phi_2$ iff $\sigma \models \phi_1$ and $\sigma \models \phi_2$

$\sigma \models \phi_1 \lor \phi_2$ iff $\sigma \models \phi_1$ or $\sigma \models \phi_2$

$\sigma \models \phi_1 U \phi_2$ iff $\exists i \leq n$ such that $\sigma^i \models \phi_2$ and $\forall j < i, \sigma^j \models \phi_1$

$\sigma \models \diamond \phi_1$ iff $\exists i$ such that $\sigma^i \models \phi_1$

2.2 Transition system

Definition 2. A weighted deterministic transition system is a tuple $\langle Q_T, q_0^T, \rightarrow_T, AP_T, L_T, w_T \rangle$, where:

1. $Q_T$ is a finite set of states;
2. $q_0^T \in Q_T$ is the initial state;
3. \(\rightarrow_T \subseteq Q_T \times Q_T\) is a deterministic transition relation;  
4. \(AP_T\) is a finite set of atomic propositions;  
5. \(L_T : Q_T \rightarrow 2^{AP_T}\) is a map giving the set of atomic propositions satisfied in a state;  
6. \(w_T : T \rightarrow \mathbb{R}^+\) is a weighting function that assigns a positive value to each transition.

We denote \(q \rightarrow_T q'\) when \(((q, q') \in \rightarrow_T)\).

**Definition 3.** A finite run \(r_T\) of a transition system \(T\) is a finite sequence of states \(r_T = q_0, q_1, ..., q_\alpha\) such that \(\forall k \in \{0, 1, ..., \alpha - 1\}\), it holds that \(q_k \in Q_T\) and \(q_k \rightarrow q_{k+1}\).

A finite run \(r_T = q_0, q_1, ..., q_\alpha\) generates a finite word \(\omega = \omega_0, \omega_1, ..., \omega_\alpha\). The infinite run generated by the finite run \(r_T\) is \(\omega\) concatenated with an infinite sequence of empty sets. Intuitively, the point of co-safe LTL is that it can be decided in finite time whether the formula holds, i.e., once it is satisfied, what happens next does not matter. We say that a finite run \(r_T\) satisfies a co-safe LTL formula \(\phi\) if and only if the word generated by \(r_T\) satisfies \(\phi\).

### 2.3 Büchi automaton

**Definition 4.** A Büchi automaton is a tuple \(B = (Q_B, q_0^B, \Sigma_B, \delta_B, F_B)\) where:

1. \(Q_B\) is a finite set of states;  
2. \(q_0^B \subseteq Q_B\) is a set of initial states;  
3. \(\Sigma_B\) is an input alphabet;  
4. \(\delta_B \subseteq Q_B \times \Sigma_B \times 2^{Q_B}\) is a non-deterministic transition relation;  
5. \(F_B \subseteq Q_B\) is a set of final states.

A run of \(B\) over an input word \(\omega = \omega_0, \omega_1, ...,\) is a sequence \(r_B = q_0^B, q_1^B, ...,\) such that \(q_0^B \in Q_B\), and \((q^k, \omega^k, q_{k+1}) \in \delta_B\) for all \(k \geq 0\). A Büchi automaton \(B\) accepts a word over \(\Sigma_B\) if and only if at least one of the corresponding runs intersects with \(F_B\) infinitely many times. For any LTL formula \(\phi\) over a set \(AP\), one can construct a Büchi automaton \(B_\phi\) with input alphabet \(\Sigma_B = 2^{AP}\) accepting all and only words over \(2^{AP}\) that satisfy \(\phi\).

### 2.4 Model-checking

Model-checking is a method used to verify that all runs \(r_T\) of a transition system \(T\) satisfy a given LTL formula \(\phi\). If it is not the case, the model-checking algorithm returns a run of \(T\) which does not satisfy \(\phi\). As a result, if we use model checking with \(\neg \phi\) we will get a run which satisfies \(\phi\).

**Definition 5.** The product of a transition system \(T = (Q, q_0, \rightarrow_T, AP, L)\) and a Büchi automaton \(B = (S, s_0, \Sigma, \rightarrow_B, F)\) with \(\Sigma \subseteq 2^{AP}\) is defined as a tuple \(A = (S_A, s_A, \Sigma, \rightarrow_A, F_A)\), where:
1. $S_A = Q \times S$ is the finite set of states;
2. $S_A^0 = q_0 \times S$ is the set of initial states;
3. $\rightarrow_A \subseteq S_A \times S_A$ is the transition relation defined as $(q_a, s_b) \rightarrow_A (q_c, s_d)$ if and only if $q_a \rightarrow_T q_c$ and $s_d \in s$ where $s$ is defined as $s_b \xrightarrow{L(q_c)} B s$;
4. $F_A = Q \times F$ is the set of final states;

This automaton accepts a run if and only if it intersects with $F_A$ infinitely many times. This product automaton can be seen as a match between the states of the transition system $T$ and the transitions of $B$.

For any run $r_A = (q_0, s_0), (q_1, s_1), \ldots$ of the product automaton $A$ which starts in an initial state, we define the projection $\gamma_T(r_A) = q_0, q_1, \ldots$ which maps $r_A$ to the corresponding run of $T$.

**Proposition 1.** If $\phi$ is a LTL formula over $AP$ and $B_\phi$ is the corresponding Büchi automaton, then the projection $\gamma_T(r_A) = q_0, q_1, \ldots$ of any accepted run $r_{A_\phi}$ of $A_\phi = T \times B_\phi$ is a run of $T$ satisfying $\phi$.

**Proposition 2.** If there is a run of $T$ satisfying $\phi$, there exists an accepting run of $A_\phi$.

The proof of those propositions can be found in [13]. Those two propositions together give a sound and complete way to address the model checking problem. The algorithm for model checking comes directly from proposition 1 and is done as follow:

1. create the Büchi automaton $B_\phi$ which accepts all and only words over that satisfy $\phi$;
2. create the product of the transitions system $T$ and the Büchi automaton $B_\phi$, $A = T \times B_\phi$;
3. look for an accepting run $r_A$ of the product automaton $A$;
4. compute the projection of this run on $T$: $r_T = \gamma_T(r_A)$.

Proposition 1 assures that $r_T$ is a run of $T$ that satisfies $\phi$.

It is easy to extend model checking to weighted transition systems by looking for the run of $A$ with the smallest cost. This allow to get the run of $T$ which satisfies $\phi$ with the smallest cost if it is a finite run.
3 Problem Formulation

In this section we will describe the set-up we are working with and how we model the robots. We will also define the cost of the robots runs we will be working with to finally formulate the problem we will aim to solve. Finally, we will show several examples to illustrate the importance of our problem.

3.1 Problem set-up

We have a team of \( n \) robots: \( r_1, ..., r_n \). The robots are in a workspace \( W \subseteq \mathbb{R}^2 \). In this workspace there are also \( m \) objects \( o_1, ..., o_m \) that the robots will want to move. We also define the empty object \( o_0 \). If a robot is not carrying any object, it carries the empty object. \( O = \{ o_0, o_1, ..., o_m \} \) is the set of all the objects.

The workspace is partitioned into \( l \) regions \( p_1, ..., p_l \) such that \( \bigcup_{i=1}^{l} p_i = W \) and \( \forall (i,j), p_i \cap p_j = \emptyset \) if \( i \neq j \). We also define the empty region \( p_0 \). If an object is carried by a robot, the object is in this empty region. We call \( P = \{ p_0, p_1, ..., p_l \} \) the set of all regions. Finally, we add an adjacency relation \( \delta \subseteq P^2 \). If there is a pair of indexes \( (i,j) \) such that \( (p_i, p_j) \in \delta \) it means that a robot which is in \( p_i \) can go to \( p_j \) without crossing any other region.

3.2 Robot model

Each robot’s \( r_i \) abstract motion capability is modelled by a deterministic transition system \( T_i = (Q, q_0^i, \rightarrow_i, AP, L, w_i) \). The state space \( Q \), the APs and the labelling function are the same for all robots.

- \( Q = W^{m+1} \times O \) is the set of states. A state \( s \) is given by a tuple \((p_{r}, p_{o_1}, p_{o_2}, ..., p_{o_m}, o_c)\) where:
  - \( p_{r} \) is the location of the robot;
  - \( \forall j \in \{1, ..., m\} \), \( p_{o_j} \) is the location of the object \( o_j \)
  - \( o_c \) is the object that the robot is carrying;

We assume here that the robot cannot carry more than one object simultaneously.

- \( q_0^i \) is the initial state.

- \( \rightarrow_i \subseteq Q \times Q \) is a transition relation. The transitions are of the following forms:
  1. \((p_{r}, p_{o_1}, p_{o_2}, ..., p_{o_m}, o_c) \rightarrow_i (p_{r}, p_{o_1}, p_{o_2}, ..., p_{o_m}, o_c)\). This transition corresponds to the robot being idle and should always be possible and costs 0, i.e., \( \forall q \in Q, q \rightarrow_i q \) and \( w_i(q,q) = 0 \).
  2. \((p_{r}, p_{o_1}, p_{o_2}, ..., p_{o_m}, o_c) \rightarrow_i (p'_{r}, p_{o_1}, p_{o_2}, ..., p_{o_m}, o_c)\) if \((p_{r}, p'_{r}) \in \delta\). This transition corresponds to the robot moving.
3. \((p_r, p_{o_1}, p_{o_2}, \ldots, p_{o_m}, o_c) \rightarrow \tau_i (p'_r, p'_{o_1}, p'_{o_2}, \ldots, p'_{o_m}, o'_c)\) if there is a \(j\) such that for every \(k \neq j\) \(p_{o_k} = p'_{o_k}\), \(p_r = p'_r\), \(p_{o_j} = p_0\), \(o_c = o_0\) and \(o'_c = o_j\). This transition corresponds to the robot picking up the object \(o_j\) which was in the same region as the robot. The robot was not carrying any other object before taking this one. This transition costs 1.

4. \((p_r, p_{o_1}, p_{o_2}, \ldots, p_{o_m}, o_c) \rightarrow \tau_i (p'_r, p'_{o_1}, p'_{o_2}, \ldots, p'_{o_m}, o'_c)\) if there is a \(j\) such that for every \(k \neq j\) \(p_{o_k} = p'_{o_k}\), \(p_r = p'_r\), \(p_{o_j} = p_0\), \(o_c = o_j\) and \(o'_c = o_0\). This transition corresponds to the robot putting the object \(o_j\) it was carrying in the region the robot is. This transition costs 1.

5. \((p_r, p_{o_1}, p_{o_2}, \ldots, p_{o_m}, o_c) \rightarrow \tau_i (p'_r, p'_{o_1}, p'_{o_2}, \ldots, p'_{o_m}, o'_c)\) if there is a \(j\) such that for every \(k \neq j\) \(p_{o_k} = p'_{o_k}\), \(p_r = p'_r\), \(o_c = o'_c\), \(p_{o_j} = p_0\) and \(p'_{o_j} = p_0\). This transition corresponds to another robot picking up the object \(o_j\).

6. \((p_r, p_{o_1}, p_{o_2}, \ldots, p_{o_m}, o_c) \rightarrow \tau_i (p'_r, p'_{o_1}, p'_{o_2}, \ldots, p'_{o_m}, o'_c)\) if there is \(j, k\) such that for every \(a \neq j, k\) \(p_{o_a} = p'_{o_a}\), \(p_r = p'_r\), \(o_c = o'_c\), \(p_{o_j} = p_0\) and \(p'_{o_j} = p_k\). This transition corresponds to another robot putting the object \(o_j\) in the region \(p_k\). This can only be done if this object was in the empty cell which means he was being carried by a robot and it was not carried by this robot \(r_i\).

The transitions 1 to 4 are driven by the robot, i.e., it is the robot that moves to another region, picks up or puts down an object; whereas the transitions 5 and 6 correspond to another robot picking up or putting down an object and are therefore uncontrollable.

- \(AP\) is the set of atomic propositions. The \(APs\) are
  - \(\pi_j = \{\text{the robot is in region } p_j\}\);
  - \(\pi_{j,k} = \{\text{the object } o_j \text{ is in region } p_k\}\);
  - \(\pi^o_j = \{\text{the robot is carrying the object } o_j\}\);

- \(L\) is a labelling function.
- \(w_i : \tau_i \rightarrow \mathbb{R}^+\) is a cost function.

### 3.3 Specifications

The specification of each robot \(r_i\) is given as a co-safe \(LTL_{-X}\) formula \(\phi_i\) over its set of atomic proposition \(AP\). This formula put constraints on the robot’s position and the position of the objects. Those \(APs\) are given by the robot transition system \(T_i\), therefore the truth value of \(\phi_i\) can be derived from only a run of \(T_i\), without looking at the runs of other robot transition systems.

### 3.4 Run of the team of robots

A finite run \(run_i\) of a robot \(r_i\) is a finite run of its transition system \(T_i\) (see Section 3.2).

**Definition 6.** A finite synchronized run of the team of robots is a tuple \((run_1, \ldots, run_n)\) where
1. \( \forall i, \text{run}_i = q_0^i, q_1^i, \ldots, q_n^i \) is a finite run of robot \( r_i \);

2. the number of states \( \alpha + 1 \) is the same in every run \( \text{run}_i, i \in \{1, \ldots, n\} \);

3. Both the following conditions are satisfied:

   - if there is a pair of indexes \( (j, k) \in \{0, \ldots, \alpha - 1\} \times \{1, \ldots, n\} \) such that \( q_k^j = (p_r, p_{o_1}, p_{o_2}, \ldots, p_{o_m}, o_c) \), \( q_{k+1}^j = (p_r', p_{o_1}', p_{o_2}', \ldots, p_{o_m}', o_c') \) and there is an index \( c \) such that for every \( a \neq b \) \( p_{o_a} = p_{o_b}, p_r = p_r' = p_{o_b}, p_{o_a}' = p_0, o_c = o_0 \) and \( o_c' = o_0 \) meaning that robot \( r_k \) is picking the object \( o_0 \);

   - then for all \( c \neq k \) it holds that \( q_c^j = (p_r, p_{o_1}, p_{o_2}, \ldots, p_{o_m}, o_c) \), \( q_{c+1}^j = (p_r, p_{o_1}', p_{o_2}', \ldots, p_{o_m}', o_c) \) and for every \( a \neq b \) \( p_{o_a} = p_{o_b}, p_{o_a}' = p_0 \) meaning that the object \( o_0 \) goes to the region \( p_0 \) for every robot while everything else stays the same.

Condition 2 is easy to achieve by adding transitions in which a robot is idling, i.e., a transition of type 1 in Section 3.2 which is always possible. Property 3 enforces that if a robot picks up or drops an object, then all the other robots do a transition which reflects this action. Those transitions are called synchronized transitions.

**Definition 7.** Consider a team of \( n \) robots, modelled as transition systems defined in Section 3.2 and their individual specifications \( (\phi_1, \ldots, \phi_n) \) over the respective AP, a synchronized run of the robots \( (\text{run}_1, \ldots, \text{run}_n) \) satisfies the specifications if \( \forall i \in \{1, \ldots, n\}, \text{run}_i \) satisfies \( \phi_i \).

### 3.5 Cost of the run of the team of robots

Now that we have defined what the runs of a robot and of the team of robots are, we can define their cost.

**Definition 8.** For a robot \( r_i, i \in \{1, \ldots, n\} \), the cost of a finite run \( \text{run}_i = q_0^i, q_1^i, \ldots, q_n^i \) is given by:

\[
\text{cost}(\text{run}_i) = \sum_{j=1}^{\alpha} w_i((q_0^i, q_1^i))
\]
The cost of a run of a robot is the sum of the cost of all its transitions. We can now define the cost of a run of the team of robots.

**Definition 9.** The cost of a finite run \( \text{run} = (\text{run}_1, ..., \text{run}_n) \) of the team of robots is:

\[
\text{cost}(\text{run}) = \sum_{i=1}^{n} \text{cost}(\text{run}_i)
\]

The cost of a finite run of the team of robots is the sum of the cost of the run of every robot.

### 3.6 Problem statement

The problem we will solve is the following:

**Problem 1.** Given models of robot motion capabilities from section 3.2 and LTL specifications as described in Section 2.1, our goal is to find a synchronized run of the team of robots which satisfies the LTL specifications of every robot with a minimal cost.

### 3.7 Examples

<table>
<thead>
<tr>
<th>p1</th>
<th>p2</th>
<th>p3</th>
<th>p4</th>
<th>p5</th>
</tr>
</thead>
</table>

Figure 1: A corridor workspace of Examples 1, 2

We will illustrate problem 1 with a few simple examples. The workspace used in the examples is a corridor divided in 5 regions as shown in figure 1. There are two robots \( r_1 \) and \( r_2 \) which want to clean some part of this corridor using a vacuum cleaner \( o_1 \). A state of the transition system of a robot is represented as \((p_r, p_o, o_1)\). We consider that both robots are identical and gives a cost 1 to all their transitions, i.e. moving to an adjacent region, picking up or putting down an object.

**Example 1 (Cooperation).** In this example, the initial states are \( q_0^1 = (p_5, p_5, o_0) \) and \( q_0^2 = (p_2, p_5, o_0) \). The first robot is in \( p_5 \), the second robot is in \( p_2 \). No robot is carrying any object and the vacuum cleaner is in \( p_5 \). The specifications are \( \phi_1 = \Box \pi_3 \) and \( \phi_2 = \Box \pi_1,1 \), i.e. robot \( r_1 \) has to go to the region \( p_3 \) and robot \( r_2 \) should bring the vacuum cleaner in the region \( p_1 \). What we would like the robots to do is that the robot \( r_1 \) brings the vacuum cleaner directly to \( p_1 \) because it is closer to the object and it will go in the good direction. This solution has a cost of 6 (1 to take the object + 4 to move to region \( p_1 \) + 1 to put down the object). If robot \( r_2 \) goes to \( p_5 \) to bring the object to \( p_1 \) itself, the cost is 12 (3 to move to region \( p_5 \) + 1 to take the object + 4 to move to region \( p_1 \) + 1 to put down the object + 2 for \( r_2 \) to go to \( p_3 \)). In this example, the robots need to cooperate to get the smallest cost.
Example 2 (Competition). In this example, the initial states are $q_0^1 = (p_5, p_3, o_0)$ and $q_0^2 = (p_3, p_3, o_6)$. The first robot is in $p_5$, the second robot is in $p_3$ and the vacuum cleaner is in the middle, in $p_3$. The specifications are $\phi_1 = \Diamond \pi_{1,5}$ and $\phi_2 = \Diamond \pi_{1,4}$, i.e., robot $r_1$ has to bring the vacuum cleaner in region $p_5$ and robot $r_2$ has to bring the vacuum cleaner in region $p_4$. Here the two robots want to bring the same object in two different locations. A good strategy would be that the second robot moves the object first because it brings it closer to where the second robot wants to put it. If robot $r_1$ moves the object first, the cost is 13.

- 2 for $r_1$ to move to $p_3$
- 1 for $r_1$ to take the object
- 2 for $r_1$ to move to $p_5$
- 1 for $r_1$ to put down the object
- 4 for $r_2$ to move to $p_5$
- 1 for $r_2$ to take the object
- 1 for $r_2$ to move to $p_4$
- 1 for $r_2$ to put down the object

If robot $r_2$ moves the object first, the cost is 9.

- 2 for $r_2$ to move to $p_3$
- 1 for $r_2$ to take the object
- 1 for $r_2$ to move to $p_4$
- 1 for $r_2$ to put down the object
- 1 for $r_1$ to move to $p_4$
- 1 for $r_1$ to take the object
- 1 for $r_1$ to move to $p_5$
- 1 for $r_1$ to put down the object

The optimal solution is that any of those robots does everything which gives a cost of 8.

Example 3 (2D). Now we will give a 2D example. It can be for example two robots moving objects in a warehouse. The initial position of the objects and the robots are given by figure 2. The specifications are $\phi_1 = \Diamond \pi_{1,21}$ and $\phi_2 = \Diamond \pi_{2,17}$, i.e., robot $r_1$ wants to bring object $o_1$ in $p_{21}$ and robot $r_2$ wants to bring object $o_2$ in $p_{17}$.

A first approach is to do a selfish run where each robot executes its specification. This run is illustrated by figure 3. Its cost is 19. The optimal solution involves cooperation with robot $r_1$ bringing object $o_2$ closer to robot $r_2$. This action increases the cost of the run of robot $r_1$ but decreases the cost of the run of the team. This run is illustrated by figure 4. Its cost is 17.
Figure 2: 2D example. $X_1$ is where robot $r_1$ wants to put object $o_1$ and $X_2$ is where robot $r_2$ wants to put object $o_2$.

Figure 3: Selfish run
Figure 4: Optimal run
4 Solution

The first step of the solution is to create a plan for each robot.

4.1 Plan’s creation

We start by defining what is a plan for a robot.

Definition 10. A plan for a robot $r_i$ is a finite sequence of states $\text{plan}_i = q^0, q^1, \ldots, q^\alpha$ such that

1. $q^0 = q_i^0$;
2. $\forall j \in \{1, \ldots, \alpha\}, q^j \in Q_i$;
3. $L_i(q^0), L_i(q^1), \ldots, L_i(q^\alpha)$ satisfies $\phi_i$;
4. given that $\forall j \in \{0, \ldots, \alpha\}$, $q^j$ is of the form $(p_j^i, p_{o_1}^j, p_{o_2}^j, \ldots, p_{o_m}^j, o^j)$, it holds that $\forall j \in \{0, \ldots, \alpha - 1\}$ there is only one $k \in \{1, \ldots, m\}$ such that $p_{o_k}^j \neq p_{o_k}^{j+1}$.

A plan gives a sequence of states that satisfies the specifications. Those states do not need to follow the transition relation of the transitions system, i.e $\forall j \in \{1, \ldots, \alpha - 1\}, (q^j, q^{j+1}) \in \delta_i$ is not required but only one object can be moved between $q^j$ and $q^{j+1}$. Because the plan does not respect the transition relation, we need to define a path between two states of the plan that respects this transition relation.

The two latest transitions of $T_i$: 5 and 6 depend on other robots movement and cannot be controlled by robot $r_i$. To decide what robot $r_i$ will do, we need a transition system which is exactly the same as $T_i$ but without those transitions which is denoted $\hat{T}_i$. $T_i$ will be used to keep track of the past transitions when $\hat{T}_i$ which do not rely on other robots will be used to decide the future transitions to do. The transition relation of $\hat{T}_i$ is denoted by $\hat{\rightarrow}_i$.

We can now talk about the path between two states in $\hat{T}_i$.

Definition 11. Given two states $(q_1, q_2) \in Q_i^2$, the path from $q_1$ to $q_2$ is defined as a finite sequence $(t_1, t_2, \ldots, t_\alpha)$ such that

1. $\forall j \in \{1, \ldots, \alpha\}, t_j \in \hat{\rightarrow}_i$;
2. there is a $q \in Q_i$ such that $t_1 = (q^1, q)$;
3. there is a $q' \in Q_i$ such that $t_\alpha = (q', q_2)$;
4. $\forall j \in \{1, \ldots, \alpha - 1\}$, there are $q, q', q'' \in Q_i$ such that $t_j = (q, q')$ and $t_{j+1} = (q', q'')$;

A path from $q_1 \in Q_i$ to $q_2 \in Q_i$ is a sequence of transitions in $\hat{T}_i$ that brings the robot from the state $q_1$ to the state $q_2$. The set of all paths from $q_1$ to $q_2$ in $\hat{T}_i$ is denoted by $\text{paths}_i(q_1, q_2)$.

The cost of a path $\text{path} = (t_1, t_2, \ldots, t_\alpha)$ for robot $r_i$ is given by the sum of the costs of its elements.

$$\text{cost}(\text{path}) = \sum_{j=1}^{\alpha} w_i(t_j)$$
To construct a plan for robot $r_i$, the first step is to construct a new transition system: an extended transition system. This will be another transition system where every transition correspond to the realisation of an atomic proposition.

**Definition 12.** An extended transition system associated to a robot model $T_i$ is a transition system $T'_i = (Q, q'_0, \sim'_i, AP, L_i, w'_i)$.

- $Q$, is the same state space as in $T_i$.
- $q'_0$ is the same initial state as in $T_i$.
- $\sim'_i \subseteq Q \times AP \times Q$ is the deterministic transition relation which is composed of a first state, an atomic proposition which describes the transition and a destination state.

1. $(p_r, p_0, p_1, ..., p_m, o_c) \xrightarrow{\alpha, \beta} (p'_r, p'_0, p'_1, ..., p'_m, o'_c)$ this transition corresponds to the robot moving without an object.
   The cost of this transition is given by the smallest cost of a sequence of transitions in $T_i$ which brings the robot from $p_j$ to $p_{\text{alpha}}$, where $j$ is defined by $p_r = p_j$.
   
   $$w'_i((q, \pi_\alpha, q')) = \min_{p \in \text{paths}(q, q')} \text{cost}_{i}(p)$$

2. $(p_r, p_0, p_1, ..., p_m, o_c) \xrightarrow{\pi_\alpha, \pi_\beta, \pi'} (p'_r, p'_0, p'_1, ..., p'_m, o'_c)$ if for every $k$, it holds that $k \neq \alpha \Rightarrow p_{\alpha_k} = p'_{\alpha_k}, p'_{\beta_k} = p_\beta, o_c \in \{o_0, o_c\}$, $o'_c = o_0$, $p_r \in P$ and $p'_r = p_\beta$, this transition corresponds to the robot moving an object. Before doing this transition, the robot could either carry object $o_\alpha$ or no object.
   The cost of this transition is the same as the cost of the transition $\pi_\beta$ plus the cost of picking up and putting down an object which is 2 (if the object was already carried by the robot, i.e, $o_c = o_\alpha$ then the additional cost is just 1).
   
   $$w'_i(((q_1^1, ..., q_1^{m+1}), \pi_\alpha, \pi_\beta, (q_2^1, ..., q_2^{m+1}))) = w'_i(((q_1^1, ..., q_1^{m+1}), \pi_\alpha, \pi_\beta, (q_2^1, ..., q_2^{m+1}))) + 2$$

   or
   
   $$w'_i(((q_1^1, ..., q_1^{m+1}), \pi_\alpha, \pi_\beta, (q_2^1, ..., q_2^{m+1}))) = w'_i(((q_1^1, ..., q_1^{m+1}), \pi_\alpha, \pi_\beta, (q_2^1, ..., q_2^{m+1}))) + 1$$

3. $(p_r, p_0, p_1, ..., p_m, o_c) \xrightarrow{\pi_\alpha, \pi_\beta} (p'_r, p'_0, p'_1, ..., p'_m, o'_c)$ if for every $k$, it holds that $k \neq \alpha \Rightarrow p_{\alpha_k} = p'_{\alpha_k}, p_r = p'_r \in P, p_{\alpha_k} \neq p_0, p'_{\alpha_k} = p_0$, $o_c = o_0$ and $o'_c = o_\alpha$, this transition corresponds to the robot picking an object.
   The cost of this transition is 1.
   
   $$w'_i((q, \pi_\alpha, q')) = 1$$
• $AP$ is the same set of atomic propositions as in $T_i$.
• $L'$ is the same labelling function as in $T_i$.

This transition system is not constrained by the adjacency relation of the regions $\delta$. Each transition correspond to the robot moving at most one object but it can go or put an object anywhere in the workspace.

The transitions of $T'_i$ are defined such that $q \xrightarrow{\pi} q' \Rightarrow \pi \in L_i(q')$. The costs of the transitions given by $w'_i$ are defined as the best-case scenario because this transition system assumes that there is no other robot to move the objects. The cost of moving an object is computed assuming the robot can pick the object on its way to its destination. The cost of the robot picking up an object is 1 because we assume that the object is in the same region as the robot so it can be picked with no additional cost. This is an ideal situation which does not take into account the position of the objects because the objects will be moved by the other robots and at this point we cannot know where they will be when the robot will want to move them.

Remark 1. For all states $(q, q') \in Q^2$, if there is a $\pi \in AP$ such that $q \xrightarrow{\pi} q'$ and $q \neq q'$, then $\pi$ is unique. We denote it $\pi_{q \rightarrow q'}$.

We will never consider the transitions $q \xrightarrow{\pi} q$ because they do not bring the robot closer to fulfilling its specification.

We can define the cost of a run $r'_i = q^0, q^1, ..., q^\alpha$ of $T'_i$ as

$$\text{cost}(r'_i) = \sum_{j=0}^{\alpha-1} w'_i((q^j, \pi_{q^j \rightarrow q^{j+1}}, q^{j+1}))$$

Proposition 3. A run $r'_i$ of $T'_i$ which satisfies the specification $\phi_i$ of robot $r_i$ is a robot plan.

Proof. 1 and 2 comes from the definition of $T'_i$. 3 is true because $r'_i$ satisfies $\phi_i$. 4 comes from the definition of the transitions of $T'_i$. □

A robot plan can thus be found by using model-checking on $T'_i$.

4.2 Synchronization

Now we have plans which give each robot a sequence of states it should go through. However, because each robot has its own plan, there may be a robot which moves an object when another robot would have wanted this object to stay where it was. To prevent that, we define for every $i \in \{0, 1, ..., \alpha\}$, where $\alpha$ is the length of the plan, the tuple $\text{lock}_i = (\text{lock}_i^1, ..., \text{lock}_i^m)$ where $\text{lock}_i^j \in \mathbb{N}$ corresponds to the number of locks added on $o_j$ in the state $q^i$. As long as the number of locks on an object is not zero, the object cannot be taken by any robot. We use this to define a locked robot plan.

Definition 13. A locked robot plan for a robot $r_i$ given its specification $\phi_i$ is defined by a sequence $(q^0, \text{lock}^0), (q^1, \text{lock}^1), ..., (q^\alpha, \text{lock}^m)$ where:
1. \( q^0, q^1, \ldots, q^\alpha \) is a robot plan;
2. \( \forall j \in \{0, 1, \ldots, \alpha\}, \text{lock}^j = (\text{lock}^j_1, \text{lock}^j_2, \ldots, \text{lock}^j_n) \in \mathbb{N}^n; \)
3. \( \text{lock}^\alpha = (0, \ldots, 0); \)
4. \( \forall j \in \{0, 1, \ldots, \alpha\} \text{ if } q^j = (p^j_0, p^j_1, p^j_2, \ldots, p^j_m, o^j) \text{ it holds that } \forall a \in \{1, \ldots, m\}, p^a_0 \neq p^{a+1}_0 \Rightarrow \text{lock}^j_a = 0; \)
5. \( \forall j \in \{0, \ldots, \alpha - 1\}, \pi = \pi_{q^j \rightarrow q^{j+1}} \) is the atomic proposition that is associated to the transition from \( q^j \) to \( q^{j+1} \), \( \forall q \in Q_i \) such that \( \exists q' \in Q_i \) and \( q' \xrightarrow{\pi_j} q \) and the robot is in the same region in \( q' \) and \( q^j \); if \( q^j, q^1, \ldots, q^{j-1}, q, q^{j+1}, \ldots, q^\alpha \) is not a robot plan, then \( q^0, q^1, \ldots, q^{j-1}, q, q^{j+1}, \ldots, q^\alpha \) violates condition 4.

To execute a plan, a robot will execute the corresponding transitions. The last property 5 guarantees that as long as the locks are respected and the transitions are done following the robot plan if one state is not exactly the same as the one defined by the robot plan (another robot has moved an object which was not locked) the specification is still fulfilled. It assures that if changing the position of an object could prevent the run to fulfil the specifications, then this object is locked and cannot be moved.

Now we define what is a run that follows a locked robot plan.

**Definition 14.** Let plan \( = (q^0, \text{lock}^0), (q^1, \text{lock}^1), \ldots, (q^\alpha, \text{lock}^\alpha) \) be a locked plan for robot \( r_i \) with respect to its specification \( \phi_i \). A run of \( r_i \) following plan is a run of \( T_i \); \( r = q^0, q^1_0, q^2_0, \ldots, q^0_n, q^1, q^2_1, \ldots, q^1_0, q^2, \ldots, q^\alpha \) where

1. \( \forall j \in \{1, \ldots, \alpha\}, \text{there exists } q \in Q \text{ such that } \pi_{q^j \rightarrow q'^j} = \pi_{q \rightarrow q'j} \text{ and the robot is in the same region in } q \text{ and } q'^j \)
2. \( \forall j \in \{0, \ldots, \alpha - 1\}, \forall k \in \{1, \ldots, \alpha_j\}, \text{ if } q^j_k = (p^j_{ik}, p^j_{ik+1}, p^j_{ik+2}, \ldots, p^j_{ik+m}, o^j_{ik+k}) \), then \( \forall a \in \{1, \ldots, m\}, \text{ it holds that } p^a_{ik} \neq p^a_{ik+1} \Rightarrow \text{lock}^j_k = 0; \) this condition also holds between \( q^j_k \) and \( q'^j+1 \) and between \( q^j \) and \( q^j_k \).

A run following a robot plan is a run that does not move locked objects and goes through some states \( q^j \) which are equivalent (they can be reached through a transition triggered by the same atomic proposition) to the states \( q^j_k \) of the plan.

The algorithm to create those locks uses the sequence of states and transitions of the plan. To create the locks, the algorithm will try to replace each of those states by all the states that can be reached by the same transition as stated in 5. It will then run this new sequence of states in the Büchi automaton corresponding to the LTL specification of the robot. If any new sequence of states is not a valid run of the Büchi automaton, it means that one of the atomic proposition that have been changed in the state prevent the system from fulfilling the specifications, so we add a lock on one of the objects which is not at the same position as in the plan. The object on which a lock is added is chosen randomly because we don’t care which object is locked, we just want to make the state which violates the specifications unreachable.

For example, let us consider a sequence of states \( (s_0, s_1, s_2) \) and a sequence of transitions \( (t_0, t_1) \). The algorithm will examine every sequence of states
(s_a, s_1, s') where s' can be reached by a transition t_1 and will lock objects so that the states violating the specification cannot be reached by a run following the locked plan. Let’s take s_2 = (p_2, p_1, p_3, o_0) for example. Suppose we replace it by s' = (p_2, p_3, p_2, o_0) and the sequence is not a valid run of the Büchi automaton. The difference between s_2 and s' is the position of the object o_1. This means that the robot needs the object o_1 to be in the region p_1 in the state s_2 to fulfil the specification. The algorithm will then put a lock on o_1.

**Algorithm 1: Synchronize**

| input | The plan plan = q^0, q^1, ..., q^a, T'_i, B_φ, |
| output | locks |
| 1 a = length(plan) |
| 2 locks = () |
| 3 for i from 0 to a do |
| 4 for j from 1 to m do |
| 5 locks(i, j) = 0 |
| 6 for i from 1 to a do |
| 7 for j from 1 to m do |
| 8 foreach state q' that can be reached by the transition π_{q_i−1→q_i} while respecting the locks do |
| 9 s'_1 is s_1 where s'_1 is replaced by q' |
| 10 if s'_1 is not a satisfying run of B_φ then |
| 11 s'_1 = (p'_1, p'_o_1, ..., p'_o_m, o'_c) |
| 12 s'_1 = (p'_1, p'_o_1, ..., p'_o_m, o'_c) |
| 13 Choose randomly an index b ∈ {k | p'_o_k ≠ p'_o_k} |
| 14 locks(i, b) = 1 |

The algorithm to create the locks is algorithm 1. The number of locks at every step is initialized at zero (lines 2-5). For every state q_i in plan it tries to replace it with every state that can be reached through the same transition (line 8). Then run the new sequence in the Büchi automaton (line 10). If one of those sequences is not a satisfying run of the Büchi, it randomly locks one of the objects that moved (lines 13-14).

### 4.3 Ordering actions

Now we consider that every robot r_i has its locked plan plan = (q^0, lock^0), (q^1, lock^1), ..., (q^a, lock^a). This plan gives every robot a sequence of actions to execute. Let’s start by formally defining an action.

**Definition 15.** An action is defined by a tuple (r_i, o_j, p_k, index) where:

- r_i is the robot that wants to do the action;
- o_j is the object robot r_i wants to move;
- p_k is the region where robot r_i wants to put object o_j.
• index is the index of the corresponding transition in the robot plan, i.e., this request corresponds to the transition from $q_i^{index-1}$ to $q_i^{index}$.

If $o_j = o_0$ this means that the robot wants to pick up object $o_j$.

An action can be associated to every transition $q \xrightarrow{\pi} q'$.

We add an index to this transition and denote it $\text{act}(q, \pi, q', \text{index})$ where $\text{index} \in \mathbb{N}$. There are 3 possible forms:

- $\text{act}(q, \pi_a, q', \text{index}) = (r_i, o_0, p_0, \text{index})$;
- $\text{act}(q, \pi_{a,b}, q', \text{index}) = (r_i, o_a, p_b, \text{index})$;
- $\text{act}(q, \pi_{a_0}^\emptyset, q', \text{index}) = (r_i, o_a, p_0, \text{index})$;

To simplify, we denote $\text{act}(q, q', \text{index}) = \text{act}(q, \pi_{a,-q'}, q', \text{index})$.

**Definition 16.** The actions associated to a robot plan $\text{plan}_i = q_i^0, q_i^1, \ldots, q_i^{\alpha_i}$ are the sequence of actions $\text{act}(q_i^0, q_i^1, 1), \text{act}(q_i^1, q_i^2, 2), \ldots, \text{act}(q_i^{\alpha_i-1}, q_i^{\alpha_i}, 1)$. Now we have the sequence of actions to be executed by every robot. The goal now is to order them so that a robot do not try to move an object which is already being carried by another robot. We will try to do it optimally. To do that, we first need to give a cost to each action.

**Definition 17.** The cost of an action $(r_i, o_j, p_k, \text{index})$ given that robot $r_i$ is in a state $q$ is given by the smallest cost of a path in $T_i$ that brings a robot to a state $q'$ $(p_i', p_{o_1}', \ldots, p_{o_m}', \alpha_i')$ where $p_{o_j}' = p_k$.

This cost of an action $\text{act} = (r_i, o_j, p_k, \text{index})$ given a robot state can be computed using model checking. The transition system used will be $T_i$ because we don’t want another robot to move the object and the LTL formula will be

1. $\Diamond \pi_a$ if $\text{act} = (r_i, o_0, p_0, \text{index})$;
2. $\Diamond \pi_{a,b}$ if $\text{act} = (r_i, o_a, p_b, \text{index})$;
3. $\Diamond \pi_{a_0}^\emptyset$ if $\text{act} = (r_i, o_a, p_0, \text{index})$;

The actions will be ordered according to the object they correspond. The actions of type 1 won’t be ordered because they do not concern an object and thus do not interfere with other robots actions.

**Definition 18.** An ordered sequence of actions is given by $\text{seq} = (\text{seq}_1, \text{seq}_2, \ldots, \text{seq}_m)$ where $\forall j \in \{1, \ldots, m\}$, $\text{seq}_j = \text{act}_{j_1}^1, \text{act}_{j_2}^2, \ldots, \text{act}_{j_n}^{\alpha_j}$ and $\forall k \in \{1, \ldots, \alpha_j\}$, $\exists (a, b, \text{index}) \in \{1, \ldots, n\} \times \{1, \ldots, l\} \times \mathbb{N}$ such that $\text{act}_{j_k}^\emptyset = (r_a, o_j, p_k, \text{index})$.

To be valid, those sequences of actions need to respect the order given by the robot plans.

**Definition 19.** A valid ordered sequence of actions is an ordered sequence of actions $\text{seq} = (\text{seq}_1, \text{seq}_2, \ldots, \text{seq}_m)$ where $\forall j \in \{1, \ldots, m\}$, $\forall (k, k') \in \{1, \ldots, \alpha_j\}^2$ if $\text{act}_{j_k}^\emptyset = (r_a, o_j, p_k, \text{index})$ and $\text{act}_{j'_k}^\emptyset = (r_a, o_j, p_{k'}, \text{index})$ it holds that index’ > index $\Rightarrow k' > k$. 

20
Definition 20. We call the action \( \text{act} = \{r, o_\beta, p_k, \text{cost}, \text{index}\} \) prior to another action \( \text{act'} = \{r', o_\beta', p_k', \text{cost'}, \text{index'}\} \) if \( \alpha' = \alpha \) and \( \text{index'} = \text{index} + 1 \).

This means that both an action and its prior will be done by robot \( r_\alpha \) and according to this robot’s plan, \( \text{act} \) should be done just before \( \text{act'} \). In a valid ordered sequence, an action should always be done after its prior.

For all \( j \in \{1, ..., m\} \), \( \forall k \in \{2, ..., \alpha_j\} \) the cost of an action \( \text{act}^k_j = (r_i, o_j, p_{k}, \text{index}) \) in an ordered sequence is a function of the robot plan and the previous action: \( \text{cost}(\text{act}^k_j, \text{plan}_{r_{\text{act}}^k_j}, \text{act}^{k-1}_j) \). The position of the robot is given by the robot plan. If the state \( q_{\text{index} - 1} \) of the robot plan is \( (p_r, p_{o_1}, ..., p_{o_m}, o_c) \) then the robot will be in \( p_r \) before doing action \( (r_i^k, o_j^k, p_{k}^k, \text{index}^k) \). The previous action \( \text{act}^{k-1}_j = (r_i^{k-1}, o_j^{k-1}, p_{k-1}^k, \text{index}^{k-1}) \) gives the position \( p_{k-1}^k \) of the object before executing \( \text{act}^k_j \). When executing an action, a robot shouldn’t touch any object except the one concerned by the action so the plan and the previous action gives the robot everything needed to compute the cost of an action.

The cost of the first action only depends on the initial situation \( \text{cost}_0(\text{act}^1_1, q_0^1) \).

For an action \( \text{act} = (r_i, o_j, p_k, \text{index}) \), we denote \( r(\text{act}) = i \) the index of the robot doing the action.

Definition 21. The cost of a valid sequence of action seq is given by

\[
\text{cost}(\text{seq}) = \sum_{j=1}^{m} \text{cost}(\text{seq}_j)
\]

where

\[
\text{cost}(\text{seq}_j) = (\text{cost}_0(\text{act}^1_j, \text{plan}_{r(\text{act}^1_j)})) + \sum_{k=2}^{\alpha_j} \text{cost}(\text{act}^k_j, \text{plan}_{r(\text{act}^k_j), \text{act}^{k-1}_j}))
\]

To find the valid sequence with the smallest cost we will examine every valid sequence of actions and compute the cost of every one of them. To do that, the algorithm will iteratively construct sequences of increasing length and their cost using Definition 21. Because there are a lot of possible combinations, the algorithm will stop at a given length to give a limited horizon solution. The algorithm is as follows.

At first, all the robots broadcast requests for all the remaining actions in their plans and the cost of doing them from the current state \( \text{cost}_0 \). Then for every action \( \text{act} = (r_i, o_j, p_k, \text{index}) \) remaining in its plan and for every request \( \text{act'} = (r_i', o_j', p_k, \text{index'}) \), every robot computes the cost of doing \( \text{act} \) with the object being in \( p_{k'} \). By adding this new cost to \( \text{cost} \) the robots already have all the costs of the sequences of actions of length 2. Those costs are saved by the robots that computed them because the cost of doing one action after another do not depend of the number of actions done before them.

Then a leader is designated for every object. Any robot can be a leader for any object. The leader can also be a computer outside of the team of robots. The leader of an object \( o_\alpha \) starts by saving all the requests on \( o_\alpha \), they form the sequences of length 1. Then the algorithm iteratively grows the length of those sequences by adding actions one by one at the end. If a sequence is found where
one action is done before its prior the sequence is removed. When adding a new action the cost to be added to the sequence can be found by looking for the cost of the sequence of the two last actions of the sequence which was computed and saved at the beginning. At the end, the robot chooses the sequence with the smallest cost among all the sequences that have been found.

The leader will execute Algorithm 2. This algorithm is given for the leader of the object \( o_\alpha \). We call \( l_\alpha \) the number of requests of the form \( \{ r_i, o_\alpha, p_k, \text{cost}, \text{index} \} \). Those requests are numbered \( req_k \) for \( k = 1...l_\alpha \).

This algorithm saves all the initial requests in a vector \( seq \) (line 1). This vector will contain all the sequences of requests computed so far. It will then add every possible action at the end of all the sequences find so far and update the cost of those sequences using the costs precomputed (lines 2-12). If a sequence is not valid, the sequence is removed (lines 7-8). The results of the algorithm is a vector containing all possible sequences of actions of length \( l_{\text{horizon}} \) that respect each robot’s plan with their cost.

Algorithm 2: Create sequences

<table>
<thead>
<tr>
<th>input</th>
<th>The requests ( req_1, ..., req_{l_\alpha} ) over the object ( o_\alpha ) and the length of the horizon ( l_{\text{horizon}} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>output</td>
<td>The sequences of actions with their costs ( seq )</td>
</tr>
</tbody>
</table>

1. \( seq = ((req_1, \text{cost}(req_1)), ..., (req_{l_\alpha}, \text{cost}(req_{l_\alpha}))) \)
2. for \( \beta \) from 2 to \( l_{\text{horizon}} \) do
3. \( seq' = () \)
4. for \( i \) from 1 to \( \text{length}(seq) \) do
5. foreach \( j \) from 1 to \( l_\alpha \) such that \( req_j \notin seq[i] \) do
6. \( (r_a, o_\alpha, p_k_j, \text{index}_j) = req_j \)
7. if there is a \( \zeta \) such that \( req_\zeta = (r_a, o_\alpha, p_k_\zeta, \text{index}_\zeta) \) with \( \text{index}_\zeta > \text{index}_j \) then
8. remove \( seq[i] \) from \( seq \)
9. else
10. \( req_\text{last} \) is the last element of \( seq[i] \)
11. Ask robot \( r_a \) the cost \( c_{\text{last},j} \) of doing \( req_j \) after \( req_\text{last} \)
12. \( seq' = (seq, req_j) \)
13. \( seq' = (seq', (seq', \text{cost}(seq') + c_{\text{last},j}) \)
14. \( seq = seq' \)

Removing deadlocks

By adding locks, it is possible to create deadlocks where at least two robots cannot finish all their actions because they prevent each other from doing their next action. To prevent the deadlocks, Algorithm 3 will examine the sequences given by Algorithm 2 and permute the actions of the sequences every time a deadlock is found.

Because this algorithm aims to stay close to the order given by Algorithm 2 which was optimal, the cost of those new sequences shouldn’t be too far from
the optimal one.

The removing deadlock algorithm is given by Algorithm 3. The algorithm first check if all the sequences are empty and return an empty sequences if it is the case (lines 1-2). Then the algorithm looks for the first object which isn’t locked (lines 3-4) and tries to do all the actions left on this object if their prior is done (lines 5-7). If all the locks are different from 0, it is a deadlock so another action has to be done (line 8). If there is a lock equal to 0, the function is called recursively to find a solution (line 9). If this recursive call finds a solution, the function appends the action it just did to the sequence found by the recursive call and returns the whole sequence as the solution (line 11). If the current action couldn’t lead to a sequence without a deadlock, the locks are reinitialised before trying the next possible action(line 12).

Algorithm 3: Remove Deadlocks

\[\text{input : The sequences of actions for each object } seq_1, seq_2, \ldots, seq_m, \text{ the list of all sequences } seq, \text{ the locks corresponding to each state. If } \]

\[\text{act brings robot } r_i \text{ from state } q^{i-1} \text{ to state } q^i, \]

\[\text{locks}(act) = \text{locks}(i, j) - \text{locks}(i, j - 1)\]

\[\text{output: } seq_1, seq_2, \ldots, seq_m\]

1 if There are no action left in any sequence then
2 return m empty sequences
3 for i from 1 to m do
4 if locks(i) = 0 then
5 foreach action act left in seq_i do
6 if The action prior to act in the corresponding robot’s plan is
7 not in any seq_j then
8 locks = locks + locks(act)
9 if There is a j such that locks(j) = 0 then
10 Call Remove Deadlocks with the updated locks and
11 seq_i, seq_j, \ldots, seq_m then
12 if Remove Deadlocks return a solution
13 seq_1', seq_2', \ldots, seq_m' then
14 return seq_1', seq_2', \ldots, (act, seq_i'), \ldots, seq_m'
15 return that a deadlock has been found
16 return that a deadlock has been found
5 Execution

In the previous chapters, we showed how a plan is computed for each robot, how locks are added to the plans and how the actions associated to the plans are ordered. In this section, we focus on the execution of those plans. Every time a robot finishes a transition, it broadcasts it to all the robots so that they can make the corresponding transition in their own transition system.

Before the execution, each robot constructs the Büchi automaton $B_{\phi_i}$ corresponding to its specification $\phi_i$ and the product automaton $A_i = T_i \times B_{\phi_i}$. This automaton will be updated every time a transition is made in $T_i$.

Then every robot will execute the actions in its plan in order. When a robot wants to execute an action $\pi$ of the form $\pi_{\alpha,j}$ or $\pi_{\alpha}^o$ it creates a Büchi automaton $B_{\phi'}$ corresponding to the LTL formula

$$\phi' = (\bigwedge_{k \in \{1, \ldots, m\} \setminus \{\alpha\}} \neg \pi_k^o) \cup \pi$$

or if it is just moving the robot:

$$\phi' = (\bigwedge_{k \in \{1, \ldots, m\}} \neg \pi_k^o) \cup \pi_\beta$$

This formula corresponds to the robot executing the action without touching any object except the one concerned by the action. This formula $\phi'$ can be modified to help another robot as shown in Section 5.1. Then the robot creates the product $B = B_{\phi_1} ' \times B_{\phi_2}'$ where $B_{\phi_1}'$ is the same Büchi automaton as $B_{\phi_1}$ except that its initial states of are given by the current states of $B_{\phi_1}$ in $A$. The finite words accepted by $B$ are the finite words that satisfy $\phi'$ from the current state and $\phi_1$ from the initial state [11]. The robot then creates the product $A' = B \times T_i$ where the initial state of $T_i$ is the current state of $T_i$. Because the current states of $B_{\phi_1}'$ were kept as the initial states of $B_{\phi_2}'$, if an accepted run of $A'$ is appended to the current run of $T_i$, the complete run will satisfy $\phi_i$. Model checking is used to find an accepted run of $A'$ which is projected onto $T_i$ to create a run $r_{A'}$. This run $r_{A'}$ is executed until $\pi$ is true or until a synchronized transition is executed. If a synchronized transition is executed, a new run is recomputed. When $\pi$ is true, the action is finished.

Once the action is finished, the robot updates the locks according to the locked plan but as long as there are locks on the object, no other robot can move it, so the robot won’t tell the other robots that its action is finished until the number of locks on the object is 0.

The online execution will be done according to algorithm 4. It looks for the next action in the robot plan (line 1-2). If this action concerns an object, the robot waits for all the actions that are planned on this object by algorithms 2 and 3 to finish (line 4). Then the algorithm use model-checking to find a path in the transition system $T_i$ that starts by fulfilling the current action but respects the LTL specifications (lines 5-9). This model-checking should take into account the path so far in the transition system $T_i$. Then the robot executes the run found by model-checking until the current action is finished (line 10). Then the locks are updated according to the locks of the current action (line 11). The robot will notify all the other robots that the action is finished but only when the number of locks on the object become 0.
Algorithm 4: Execution

\textbf{input}: The robot plan }$P$\text{ and the order in which actions should be done, ans the ltl specification }$\phi$

\textbf{while} There is an action left in }$P$\textbf{ do

\hspace{1em} \textbf{act} is the current action

\hspace{2em} \textbf{if} \text{ act is of the form }$\pi_{i,j}$\textbf{ or }$\pi_{i}^{o}$\textbf{ then}

\hspace{3em} Use model-checking to find a run of the automaton to do

\hspace{4em}$\phi \land \left( ( \bigwedge_{k \in \{1, \ldots, m\} \setminus \{j\}} \neg \pi_{k}^{o} ) U act \right) \right)$

\hspace{4em} Wait for the action before }$\text{ act}$\text{ on the object }$o_{i}$\text{ to finish

\hspace{2em} \textbf{else}

\hspace{3em} \textbf{act} is of the form }$\pi_{i}$\textbf{}

\hspace{4em} Use model-checking to find a run of the automaton to do

\hspace{5em}$\phi \land \left( ( \bigwedge_{k \in \{1, \ldots, m\}} \neg \pi_{k}^{o} ) U act \right) \right)$

\hspace{4em} Execute this run until }$\text{ act}$\text{ is satisfied

\hspace{2em} \textbf{if} \text{ a synchronized transition is done by an other robot then}

\hspace{3em} Update the current state

\hspace{3em} Recompute the run

\hspace{3em} Update the locks

\hspace{3em} \textbf{foreach} lock which are equal to }$0$\textbf{ do

\hspace{4em} Broadcast that the previous actions on the corresponding object are finished
5.1 Cooperation

Finally, we want to address the problem of cooperation, i.e., how to make a robot moves an object instead of another robot because is reduces the cost. Every time a robot \( r \) starts a new action \( \pi_{i,j} \) or \( \pi'_i \) where an object needs to be moved, it will broadcast \( \{ \pi_{i,j}, \text{cost} \} \) or \( \{ \pi_{i,j}, \text{cost} \} \) to all the robots, where cost is the cost of executing this action for the robot. The other robots will try to help robot \( r \) by executing the action for it or by bringing the object closer. We call \( p(o_r) \) the index of the position of object \( o_r \), i.e., object \( o_r \) is in region \( p_p(o_r) \). Each robot will then try to help robot \( r \). To help robot \( r \), if robot \( r' \) is doing an action \( \pi = \pi_{r',j'} \) it will compute the cost of doing \( \Diamond \pi_{r',j'} \land \Diamond \pi_{p(o_r)} \). If robot \( r' \) is doing an action \( \pi' \) it will compute the cost of doing \( \Diamond \pi_{p(o_r)} \land \Diamond \pi' \).

We call \( \text{diff} \) the difference between the cost of this new action and the cost of the action without helping the other robot. If \( \text{diff} > \text{cost} \), it costs more for the helping robot \( r' \) to reach the object than it costs robot \( r \) to do its action. Robot \( r' \) will tell that it cannot help. If robot \( r' \) can help, it will try to compute iteratively the cost of moving the object one region further in the direction of \( p_j \) if \( r \) starts \( \pi_{i,j} \) or in the direction of \( r' \)'s position if \( r \) starts \( \pi'_i \) until the global cost of the actions increases. We assume that if moving the object one step further increases the global cost, it won’t decrease if we move it further. The region which gives the smallest cost is \( p_k \).

Robot \( r' \) will then execute its new action by satisfying

\[
\phi' = \left( \bigwedge_{\alpha \in \{1,\ldots,m\} \setminus \{i,i'\}} \neg \sigma^o_{\alpha} \right) \mathcal{U} (\pi_{i',j'}) \land \left( \bigwedge_{\alpha \in \{1,\ldots,m\} \setminus \{i,i'\}} \neg \sigma^o_{\alpha} \right) \mathcal{U} (\pi_{i,k})
\]
or

\[
\phi' = \left( \bigwedge_{\alpha \in \{1,\ldots,m\} \setminus \{i,i'\}} \neg \sigma^o_{\alpha} \right) \mathcal{U} (\pi_{i,k}) \land \left( \bigwedge_{\alpha \in \{1,\ldots,m\} \setminus \{i,i'\}} \neg \sigma^o_{\alpha} \right) \mathcal{U} (\pi^o_i)
\]
or

\[
\phi' = \left( \bigwedge_{\alpha \in \{1,\ldots,m\} \setminus \{i\}} \neg \sigma^o_{\alpha} \right) \mathcal{U} (\pi_{i,k}) \land \left( \bigwedge_{\alpha \in \{1,\ldots,m\} \setminus \{i\}} \neg \sigma^o_{\alpha} \right) \mathcal{U} (\pi^o_i)
\]

We consider that the order in which the actions are done is not important. This can be extended to help several robots by moving \( \beta \) objects: \( i_1, i_2, \ldots, i_\beta \) to the regions \( k_1, k_2, \ldots, k_\beta \) by doing :

\[
\phi' = \left( \bigwedge_{\alpha \in \{1,\ldots,m\} \setminus \{i_1,i_2,\ldots,i_\beta\}} \neg \sigma^o_{\alpha} \right) \mathcal{U} (\pi_{i',j'}) \land \left( \bigwedge_{\alpha = 1}^{\beta} \left( \bigwedge_{\alpha \in \{1,\ldots,m\} \setminus \{i_1,i_2,\ldots,i_\beta\}} \neg \sigma^o_{\alpha} \right) \mathcal{U} (\pi_{i_\alpha,k_\alpha})
\]
or

\[
\phi' = \left( \bigwedge_{\alpha \in \{1,\ldots,m\} \setminus \{i_1,i_2,\ldots,i_\beta\}} \neg \sigma^o_{\alpha} \right) \mathcal{U} (\pi_{i,k}) \land \left( \bigwedge_{\alpha = 1}^{\beta} \left( \bigwedge_{\alpha \in \{1,\ldots,m\} \setminus \{i_1,i_2,\ldots,i_\beta\}} \neg \sigma^o_{\alpha} \right) \mathcal{U} (\pi_{i_\alpha,k_\alpha})
\]

26
The number of iterations of this algorithm can be limited to make it faster. Instead of looking for the cost of moving the object one step further until the cost increases, we can stop after a fixed number of steps.

Algorithm 5: Try to help

input : The current action of the robot $\pi$ and the action of the robot we want to help $\pi_{i,j}$ and its cost $cost$

output: The action to do by both robots $act_1$ and $act_2$

1 Use model-checking to find the cost $c$ of doing $\pi \land \pi_{p(\omega)}$

2 if $c \leq cost$ then

3 No help possible

4 else

5 $cost_{ref} = cost$

6 $cost_{new} = 0$

7 $act_1 = 0$

8 $act_{new}^1 = 0$

9 $act_2 = 0$

10 $act_{new}^2 = 0$

11 while $cost_{new} \leq cost_{ref}$ do

12 $cost_{ref} = cost_{new}$

13 $act_1 = act_{new}^1$

14 $act_2 = act_{new}^2$

15 Try to move the object one step further in the direction of $p_j$, this action is $act_{new}^1$

16 Ask the other robot what is the cost of doing $\pi_{i,j}$ with the new position of the object, this action is $act_{new}^2$

17 $cost_{new}$ is the sum of the cost of the new actions of the two robots

18 if the new position of the object is $p_j$ and $cost_{new} \leq cost_{ref}$ then

19 $cost_{ref} = cost_{new}$

20 $act_1 = act_{new}^1$

21 $act_2 = act_{new}^2$

22 Stop the function

6 Illustration

This framework was implemented in C++. The robots are represented by gray squares. The object is represented by a circle. The circle is blue when the object is on the floor, and it is red when the object is taken by a robot. The figures 5
to 9 shows the results for example 1. In this case, we get the optimal solution with robot 1 bringing the object to $p_1$.
The plan found by robot 1 is $(p_1, p_1, o_0)$ which corresponds to the ap $\pi_{1,1}$ and the corresponding action is $act_1 = (r_1, o_1, p_1, 1)$. The plan found by robot 2 is $(p_3, p_3, o_0)$ which corresponds to the ap $\pi_{3}$ and the corresponding action is $act_2 = (r_2, o_0, p_3, 1)$.
The cost of $act_1$ from the initial state is 8 (3 to go to $p_5 + 1$ to grab the object + 4 to go to $p_1 + 1$ to put down the object).
The plan for object 1 will be $(act_1)$. There cannot be any deadlock because there is only one object (and only one action).
Then the robots start executing their actions. Robot 1 starts $act_1$ when robot 2 starts $act_2$. Robot 1 starts by asking for help. Robot 2 can help robot 1 because it costs 0 for it to go to $p_5$ where the object is.
The cost for robot 2 to move the object to $p_4$ and fulfill its specification is 4 (1 to grab the object + 1 to go to $p_4 + 1$ to put it down + 1 to go to $p_3$) and reduces the cost for robot 2 by 1. The total cost is then 11.
The cost for robot 2 to move the object to $p_3$ is 4 and reduces the cost for robot 1 by 2. The total cost is then 10. The total cost has not increased so robot 1 will try to move the object one step further.
The cost for robot 2 to move the object to $p_2$ is 5 and reduces the cost for robot 1 by 3. The total cost is then 10. The total cost has not increased so robot 1 will try to move the object one step further.
The cost for robot 2 to move the object to $p_1$ is 6 and reduces the cost for robot 1 to 0. The total cost is then 6. The object cannot be moved any further so robot 2 will bring the object to $p_1$ and robot 1 will do nothing.
So robot 2 executes $\diamondsuit \pi_3 \wedge \diamondsuit \pi_{1,1}$. The path found using model-checking is:
$(p_5, p_5, o_0), (p_5, p_0, o_1), (p_4, p_0, o_1), (p_3, p_0, o_1), (p_2, p_0, o_1), (p_1, p_0, o_1), (p_1, p_1, o_0)$. Robot 2 takes the object, goes to $p_4$, then to $p_3$, then to $p_2$ and to $p_1$ where it drops the object.
The specifications of both robots are fulfilled so they shut down.
Figure 5: Example 1, step 1

Figure 6: Example 1, step 2

Figure 7: Example 1, step 3

Figure 8: Example 1, step 4

Figure 9: Example 1, step 5
The figures 10 to 15 shows the results for example 2. In this case, we don’t get the optimal solution where one robot does everything because in this implementation, a robot shuts down as soon as its specification is fulfilled and stops trying to help.

The plan found by robot 1 is \((p_5, p_5, o_0)\) which corresponds to the ap \(\pi_{1,5}\) and the corresponding action is \(act_1 = (r_1, o_1, p_5, 1)\). The plan found by robot 2 is \((p_4, p_4, o_0)\) which corresponds to the ap \(\pi_{1,4}\) and the corresponding action is \(act_2 = (r_2, o_1, p_4, 1)\).

The cost of \(act_1\) from the initial state is 6 (2 to go to \(p_3 + 1\) to grab the object + 2 to go to \(p_5 + 1\) to put down the object).

The cost of \(act_2\) from the initial state is 5 (2 to go to \(p_3 + 1\) to grab the object + 1 to go to \(p_4 + 1\) to put down the object).

After \(act_2\), the object will be in \(p_4\) and robot 1 will still be in \(p_5\). The cost of \(act_1\) after \(act_2\) is 4 (1 to go to \(p_4 + 1\) to grab the object + 1 to go to \(p_5 + 1\) to put down the object).

After \(act_1\), the object will be in \(p_5\) and robot 2 will still be in \(p_1\). The cost of \(act_2\) after \(act_1\) is 7 (4 to go to \(p_5 + 1\) to grab the object + 1 to go to \(p_4 + 1\) to put down the object).

The cost of doing \(act_1\) and then \(act_2\) is 6 + 7 = 13; the cost of doing \(act_2\) and then \(act_1\) is 5 + 4 = 9. The plan for object 1 will therefore be \((act_2, act_1)\). There cannot be any deadlock because there is only one object.

Then the robots start executing their actions. Robot 2 starts \(act_2\) when robot 1 waits for \(act_2\) to finish to execute. Robot 2 starts by asking for help. Robot 1 cannot help robot 2 because if it gives the object to robot 2, the cost is increased and if robot 1 does all the action, the cost is the same.

So robot 2 executes \(act_2\). The path found using model-checking is: \((p_1, p_3, o_0), (p_2, p_3, o_0), (p_3, p_5, o_0), (p_3, p_0, o_1), (p_4, p_0, o_1), (p_4, p_4, o_0)\). Robot 2 goes to \(p_1\), then to \(p_3\), picks up the object, goes to \(p_4\) and drops the object.

In this implementation because the specification of robot 2 is fulfilled it shuts down. Then robot 1 starts \(act_1\). There is no robot left to help. The path found using model-checking is: \((p_5, p_4, o_0), (p_4, p_4, o_0), (p_4, p_0, o_1), (p_5, p_0, o_1), (p_5, p_5, o_0)\). Robot 1 goes to \(p_4\), picks up the object, goes to \(p_5\) and drops the object. Then the specification of robot 1 is fulfilled and it shuts down.
<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 10: Example 2, step 1

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 11: Example 2, step 2

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 12: Example 2, step 3

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 13: Example 2, step 4

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 14: Example 2, step 5

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 15: Example 2, step 6
7 Discussion

The solution proposed by this thesis has advantages and disadvantages. This solution is flexible, if a robot disconnects from the team because it needs to recharge for example the plan and the actions of the other robots will not change, the only thing that needs to be recomputed is the order of the actions. The same thing is true if a robot is added to the team.

It is also robust if an object is moved not by a robot. If this happens, it can be modelled by a synchronized transition of a robot not in the team. All robots will perform a transition to a state where this object is in region $p0$. And then we can ask the robots to use their sensors to detect the objects every time they move to a region and perform a corresponding synchronized transition if an object is missing or appearing in that region.

The system of locks is not flexible and locking the objects randomly can prevent the team of robots of finding a solution even if a solution exists. Another disadvantage with the locks is that locking the initial states can create a deadlock before any robot does an action which cannot be removed by the remove deadlocks algorithm. However, not locking the initial state can cause the specifications of the form $\pi_{i,j} U \phi$ to be violated by another robot, because we cannot lock the object $o_i$ in the initial state so it can be moved before $\phi$ is satisfied. An alternative could be to remember all the states that violate the plan that were found during the creation of the locks and use them to order the actions.

Another factor that makes this framework lacks flexibility is the fact that the plan of each robot which is computed without taking into account the other robots specifications and is never recomputed. As a consequence, if a robot has a specification which contains something of the form $\phi_1 \Rightarrow \phi_2$, this robot’s plan will be to do nothing if its specification does not contains something equivalent to $\Diamond \phi_1$. But if another robot makes $\phi_1$ become true, the first robot will not be able to react and will conclude that it cannot fulfil its specification because its plan is not valid any more. A way to address this could be to occasionally recompute the plans.

A major restrictive factor of the solution is the size of the transition systems. The size of the state space $Q$ is $O(l^{m+1}m)$. It is exponential in the number of objects. As a result, creating the transition systems described in this thesis with only 2 objects uses a lot of memory and is only possible for workspaces with few regions. In addition, because of the high number of states, exploring the transition systems to find paths also takes a lot of time.
8 Conclusions and Future Work

The future work includes the improvement of the proposed algorithms from various perspectives, and particularly of the flexibility of the lock system. In the execution of each action, a path is found to satisfy the LTL formula associated to the action but if a synchronized transition happens, the run is recomputed. Another future direction will involve investigation of the ways to update the current run without recomputing it from scratch. Most of the time, the recomputed run is exactly the same as the previous one.

In this thesis, the specifications are limited to co-safe LTL formulas to have a finite number of actions which prevents any robot from waiting forever to do its action. Extension to full LTL is another possible research direction, e.g., by looking for a plan in a prefix-suffix form and applying what is described in this thesis to the prefix and to the suffix.

Finally, a last possible research direction is to look for how and when a robot can recompute or refine its plan.
References


