UNIVERSITY OF GENOVA

PHD PROGRAM IN BIOENGINEERING AND ROBOTICS

Task planning and allocation for multi-agent collaborative robot systems

by

Hossein Karami

Thesis submitted for the degree of *Doctor of Philosophy* (34° cycle)

May 2022

Professor Fulvio Mastrogiovanni — Supervisor

Professor Giorgio Cannata — Head of the PhD program

Dibris

Department of Informatics, Bioengineering, Robotics and Systems Engineering

It is hard to accomplish a PhD without the support of people about whom you care and they unconditionally care about you. It is also hard for me to forget them, since they always pass through my thoughts and the memories with them, make me smile. Yet, it is also hard to write and appreciate the support they gave me throughout my life, during my PhD or even for short periods of their presence. Nevertheless, i heartfully dedicate this thesis to them to partially express my love to them.

I dedicate this thesis to:

My parents: who have never stopped their love and support. They always have been present in my life, to educate me and give meanings to my existence. Despite the large physical distance among us, their continuous love and care has always pushed me to move forward to find objectives of my life as a human being. You both are the only reasons of my life, love you so much.

My brothers and the unique sister : who have always sacrificed for their younger brother. In many cases, they preferred my convenience over their comfort, not only because we share the same blood, but also to show that love among siblings exists and is mainly expressed non-verbally. Thanks for being the pillars of my growth, and giving beautiful nephews to me.

My Klelja: my love and my life. Her presence has positively formed and organised my life. She and her continuous love are special and priceless for me. She has always been present even in most difficult moments of life. *Amore mio*, Thanks for being with me, i love you enormously. I would also like to appreciate the supports and love received from her family, they are a part of my family as well. You all matter to me.

Fulvio Mastrogiovanni: it was a honor for me to get to know him. He is an intelligent person and a kind man. He has also been a valuable supervisor and mentor for my PhD program. As a supervisor, he was flexible enough to allow me to explore new areas and determinant enough to guide me toward objectives of my PhD. Thanks Fulvio for everything.

My colleagues and lab-mates, Yusha Karim, Antony Thomas, Alessandro Carfi, Mohamad Allameh and Simone Maccio. I was lucky to get to know them and work with them in the

same lab. They inspired me a lot, both in life and research. It was an achievement for me to make friends out of colleagues.

My adorable friends, Saeed Yousefvand, Reza Kamali and Nima Arteghzadeh. I got to know them since my undergraduate studies time, and our friendship has exponentially strengthened over time. Many other friends such as Saber Mohammadi, my friend and lockdown time flatmate, and more others are obviously in my heart and their names are omitted to shorten this list. However, you all have always been important for me and you can always count on me as i do.

Declaration

I hereby declare that except where specific reference is made to the work of others, the contents of this dissertation are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other university. This dissertation is my own work and contains nothing which is the outcome of work done in collaboration with others, except as specified in the text and Acknowledgements. This dissertation contains fewer than 65,000 words including appendices, bibliography, footnotes, tables and equations and has fewer than 150 figures.

Hossein Karami
May 2022

Acknowledgements

Although it was tough to imagine to reach this point, but hard work pays off. Doing a PhD is a nightmare, it's not only the matter of passion for doing research, it is a vital step in life that changes your attitude toward the universe in which you are lost and need to position yourself inside it. I think for me it was necessary to follow a PhD in robotics engineering. It helped me a lot to grow and open my eyes to perceive better, and also to realize who i am and to better realize what i want. However, it was not all about fear and confusion, hope was always there. The PhD program also made me grow in technical aspects. It taught me how to be independent, and a team player at the same time in accomplishing pure research.

I was not alone in this academic achievement, My professor and colleagues also had key roles. I would also like to thank university of Genoa, in particular, Department of DIBRIS and all of the administration staff who assisted me with all paper works and bureaucratic stuff for the whole period of my PhD. My PhD was financially supported by *Regione Liguria*, i am grateful to them for their support that allowed me to accomplish this PhD without economical concerns.

I regret that my PhD plans did not work as i was expecting, and could not reach objectives of my PhD as i had planned. COVID-19, beside all life related issues, was the major obstacle that prevented me from achieving what i really was looking for. Anyway, this is life, and we need to adapt ourselves to it and change our plans and expectations accordingly.

Abstract

In this thesis we address the problem of human robot interaction in industrial environments from collaboration perspective. This thesis, in particular, focuses on introducing novel frameworks for coordination of heterogeneous teams made of humans and robots that collaboratively aim to reach to a common goal.

In the last decade, robots has received enormous attentions for being employed in both industrial environments and workplaces. Thin is mainly because of a number of reasons: (I) the shift of mass production industries toward autonomous industry units, (II) huge amount of financial and scientific investments on robotics, and (III) substitution of humans with robots to accomplish hazardous and stressful tasks. However Due to the limited cognitive knowledge and reasoning of the robots in accomplishing complex operations, they still are not able to operate in a fully autonomous fashion and independently from their human counterparts. Therefore presence of human operators, as a complementary counterparts, in workplaces becomes fundamental for robots to become utterly safe, reliable and operative. The goal of this thesis is to design and implement a framework whereby humans and/or robots can together play a complementary role, while applying their individual skills to accomplish a task.

Human-robot collaboration (HRC) is defined as the purposeful interaction among humans and robots in a shared space, and it is aimed at a common goal. The design of such framework for HRC problems, requires to satisfy many requirements from which *flexibility*, *adaptability* and *safety*, are the primary characteristics of such framework.

In this thesis we mainly focus on multi-agent robot systems task allocation and planning. We consider two main aspects in defining our objectives: on one hand we investigate on HRC, and implement alternative frameworks to model and study collaborations in industrial scenarios considering various roles of humans in coordination and collaboration with robots. On the other hand, the presence of humans is neglected and it is assumed that robots are able to fully precept the environment independently from human cognitive support, as this can be the case of future where *Artificial intelligence* might substitute the skills of humans.

To model a HRC scenario, a smart framework is required to start, coordinate and terminate

a collaborative process. This framework, in particular has to be aware of agents and their types, determine their responsibilities and roles, be aware of their physical structure, define the logical relationship among the agent and handle the collaboration process fluently.
In this thesis, to address the framework described above, we propose different frameworks and evaluate their effectiveness in solving HRC problems. To formulate task planning and allocation problem , we introduce and implement three variants of AND/OR graphs, namely, *c-layer AND/OR graphs*, *Branched AND/OR graphs*, and *Iteratively deepened AND/OR graphs*. The first two variants aim at addressing the problem of *task allocation* among humans and collaborative robots in object defect inspection (ODI) scenarios in HRC context. Instead, the third variant targets *Task and motion planning* (TAMP) problems for heterogeneous robots. TAMP problems, compared to HRC problem, is not only responsible for allocating task among agents at higher-level, but also at lower level, it plans motions for agents and ultimately, interconnects higher levels of task planning to lower levels of motion planning and control to achieve a complete planning framework. To validate the applicability and scalability of our proposed frameworks, we design and implement various real-world and simulation experiments and we also evaluate their effectiveness in terms of achieving desired objectives, and quantitatively with other available methods in the literature.

Table of contents

List of figures

List of tables

Nomenclature

Acronyms / Abbreviations

AOG	AND/OR Graph
COBOTS	Collaborative Robots
EG	Empty Graph
GMM	Gaussian Mixture Modelling
HAR	Human Activity Recognition
HRC	Human-Robot Collaboration
KB	Knowledge Base
KTG	Kinesthetic Teaching Graph
MP	Motion Planner
ODD	Object Defect Detection
CONCHRC	Concurrent Human-Robot Collaboration
ODI	Oject Defect Inspection
TAMP	Task and Motion Planning
PDDL	Planning Domain Definition Language
TMPI	Task Motion Planning Interface
TP	Task Planner

Chapter 1

Introduction

1.1 Multi-agent task planning and allocation

The presence of industrial robots in factories and manufacturing work places exposes high risk danger to human beings present in the environment. Therefore, these robots are used to operate in fenced cages or segregated spaces where the presence of humans is not allowed. To the vast majority of the occasions, industrial robots are produced and programmed to accomplish some specific, repetitive and limited robotic tasks and their reusability, usually comes at the cost of producing new robots. Inevitably light-weight collaborative robots (Cobots), in recent decade, were introduced to ensure both human safety in operation work places and ensure their reusability.

Despite all advantages that cobots, as complementary counterparts to humans, bring with themselves, they entail many challenges. One of the main challenges is raised due to the *coexistence* of a team of counterparts composed of humans and cobots. This challenge emerges the demand for a framework that is able to determine *roles* and *rules* among all the participants, or in other words, agents, for entire a collaboration process. In this thesis we mainly focus on addressing this challenge by introducing and implementing the state of the art solutions.

Multi-agent systems, share some common features such as, i) incomplete information or capabilities for solving the problem in each agent, ii) no system global control, iii) decentralized data, and iv) asynchronous computation [89]. However multi-agent robot systems as a sub-class of multi-agent systems, add their own features. Multi-agent robot systems are composed of heterogeneous robots and human agents and each agent exposes their own capabilities and limitations differently against each of other agents.

Therefore, the demand for developing a framework that is able to coordinate and monitor

agents becomes an essential core for a collaborative multi-agent robot systems. This framework should be able to properly allocate tasks among agents dynamically to grant an efficient collaboration [40].
In this thesis we focus on multi-agent robot systems task planning and allocation. In particular we consider *Human-robot collaboration* (HRC) and *Task and motion planning* (TAMP) aspects of multi-agent systems. In HRC, human and robot agents together follow a common goal under supervision of framework, that dynamically allocates tasks to agents and gives a sort of freedom and flexibility to human agents at *decision making level*, whereas in TAMP, human agents are absent and heterogeneous robots are dynamically allocated for task execution.
Frameworks used in HRC and TAMP problems follow a similar *system architecture*. The system architecture embeds various levels of abstraction to grant a natural and efficient task allocation and planning. Later in chapters 4, 5 and 7 we will discuss them in detail.

1.2 Motivation

Despite significant results achieved in different research fields of robotics, robots are far from substituting human workers because of their limited cognitive capabilities, non-reliable perception systems, and non-adaptive decision-making processes. For these reasons, they often perform simple specialized tasks, and they have to rely on human help for more complex ones. The Industry 4.0 paradigm envisions a close relationship between robots and human operators to overcome these limitations and achieve higher shop-floor flexibility. In this context, robots should both operate autonomously and collaborate with operators.

The execution of robotic tasks needs a plan detailing all the necessary actions and their execution order. This plan, in collaborative scenarios, should be shared with an operator and, to favour an intuitive and flexible human-robot collaboration, the system should adapt it to the operator's actions and sudden variations of the task.

In many industries and workshops due to physical and chemical characteristics of products, identifying defects in inspection process jeopardizes the workers' safety and we argue that collaboration between an experienced human operator and a robot may lead to higher rates in defects spotting, overall productivity, and safety [25, 62].

HRC is defined as the purposeful interaction among humans and robots in a shared space, and it is aimed at a common goal. A natural collaboration requires a robot to perceive and correctly interpret the actions (as well as the intentions) of other humans or robots [2, 86].

State of the art robotic solution together with skilled inspectors in a HRC framework provide fast, accurate, safe and repeatable way of inspection ensuring that final product is free of defects. Maneuverability, accuracy and strength of Cobots added to the human experience and skills lead to maximising product quality and minimising final cost.

This reinforces the safety, efficiency, and performance and lowers the psychological discomfort and stress in the workplace [25, 62].

1.3 Thesis contribution

In this thesis we introduce and implement three variants of AND/OR graphs, namely, *c-layer AND/OR graphs* that targets concurrent multi-agent task allocation and planning, *Branched AND/OR graphs*, that enables a flexible and adaptable HRC, and *Iteratively deepened AND/OR graphs* that tackles the issue of infinite horizon task planning. The first two are aimed at addressing the problem of *task allocation* among humans and cobots in object defect inspection (ODI) scenarios in HRC context. Instead for TAMP problem, we propose a framework that is not only responsible for allocating task among agents at higher-level, but also at lower level, it is able to plan motions for agents and ultimately, relates low and high levels of planning to form a complete TAMP framework. The TAMP framework adopts the third variant of AND/OR graph.

However, primary contributions of thesis thesis are listed as follows:

C1 Develop and implement *c*-layer AND/OR graphs to model concurrency problem in multi-agent task allocation and planning in robotic problems.

C2 Design and implement of CONCHRC system framework to embed *c*-layer AND/OR graphs to representation layer and also integrate HRC modules for multi-agent robot systems task allocation and planning.

C3 Implement an instance of HRC scenario for ODI with four robot agents and one human agent to validate *c*-layer AND/OR graphs within CONCHRC framework.

C4 Develop and implement of branched AND/OR graphs to model an adaptable and flexible HRC process to extend the flexibility of human agent *decision making* and plan modification at online phase.

C5 Design and implement a framework to accommodate branched AND/OR graphs as task representation module in HRC framework.

C6 Implement an instance of HRC scenario for a ODI scenario with two robot agents and one human agent to validate applicability and performance of branched AND/OR graphs.

C7 Develop and implement iteratively deepened AND/OR graphs to model a network of AND/OR graphs with unknown depth.

C8 Design and implement TMP-IDAN system framework to implement task and motion planning problems in robotics with unknown horizon.

C9 Implement an instance of TAMP scenario with single robot, for target object retrieval from a cluttered table top with unknown number of occluding objects, to validate iteratively deepened AND/OR graphs within TMP-IDAN framework.

C10 Implement an instance of TAMP scenario with multiple robots, for target objects retrieval from a shared cluttered table top with unknown number of occluding objects, to validate iteratively deepened AND/OR graphs within TMP-IDAN framework.

C11 Implement TMP-IDAN for five TAMP benchmarks to validate applicability and scalability of iteratively deepened AND/OR graphs for all task and motion planning problems in robotics.

As a result of aforementioned contributions, the following scientific papers are published or are ready to submit in both national and international conferences and journals.

- H. Karami, K. Darvish and F. Mastrogiovanni, "A Task Allocation Approach for Human-Robot Collaboration in Product Defects Inspection Scenarios," 2020 29th IEEE International Conference on Robot and Human Interactive Communication (RO-MAN), 2020, pp. 1127-1134, doi: 10.1109/RO-MAN47096.2020.9223455.
- H. Karami, A. Carfì and F. Mastrogiovanni, "Branched AND/OR Graphs: Toward Flexible and Adaptable Human-Robot Collaboration," 2021 30th IEEE International Conference on Robot & Human Interactive Communication (RO-MAN), 2021, pp. 527-533, doi: 10.1109/RO-MAN50785.2021.9515512.
- Thomas, A., Karami, H., & Mastrogiovanni, F. (2020). Iterative AND/OR Graphs for Task-Motion Planning. In Italian Conference on Robotics and Intelligent Machines (IRIM).

- Karami, H., Carfi, A. & Mastrogiovanni, F. (2020). An Integrated Approach to Represent and AdaptHuman-robot Collaborative Tasks. In Italian Conference on Robotics and Intelligent Machines (IRIM).
- Karami, H., Thomas, A. and Mastrogiovanni, F., 2021. A Task and Motion Planning Framework Using Iteratively Deepened AND/OR Graph Networks. arXiv preprint arXiv:2110.04089. Ready to submit.
- Karami, H., Thomas, A. and Mastrogiovanni, F., 2021. Benchmark Evaluation in Task and Motion Planning Using Iteratively Deepened AND/OR Graph Networks. Ready to submit.

1.4 Thesis structure

The remainder of this thesis is organized through 9 chapters as follows:

- **Chapter 2** Introduces plain AND/OR graphs as the essential core to task allocation framework and presents it's formalization. In this chapter we formalize AND/OR graphs and explain their underlying structure that form the basis of all the three variants as major contributions of this thesis.
- **Chapter 3** Familiarizes the readers to HRC concepts specifically for ODI scenarios, where a team made of a human, one mobile manipulator and a dual arm robot manipulator, collaboratively inspecting the quality of products.
- **Chapter 4** Presents CONCHRC framework and it's architecture that we adopted for ODI process. *c*-layer AND/OR graphs are presented and implemented to model concurrency for collaborations among all agents. An instance of ODI scenario is introduced and implemented and ultimately qualitative and quantitative results are discussed.
- **Chapter 5** Introduces branched AND/OR graphs, and implements a flexible and adaptable HRC framework that allows the human operators to interactively interrupt high level of planning and kinesthetically teach low level of motions to cobots.
- **Chapter 6** Introduces TAMP problems in robotic problems and positions our method in the literature to better compare it with existing methods in terms of efficiency,scalability and performance.

- **Chapter 7** Introduces TMP-IDAN framework used for implementing TAMP problems with unknown number of objects for rearrangements and presents iteratively deepened AND/OR graphs as the core of the task representation of the architecture. Finally, two models of TMP-IDAN, i.e., single and multiple agent are implemented and validated.
- **Chapter 8** Implements five TAMP benchmarks, that are defined to assess any TAMP solution. Iteratively deepened AND/OR graphs are used to address the TAMP benchmarks, and superiority of our method over other existing methods is discussed.
- **Chapter 9** Concludes the whole of this thesis and discusses future work suggestions, improvements and directions for interested readers.

Chapter 2

AND/OR graphs

2.1 Definition and Formalisation

In this chapter we introduce AND/OR graphs and formalize them for the readers. We mainly adopt AND/OR graphs in our work to to represent high level of task representation in HRC and TAMP problems. AND/OR graphs are easy to comprehend and also to implement. This makes them popular among all task representation methods such as, PDDL-based planners or Markov Decision Process (MDP) planners. Later in this thesis we introduce and implement three different variants of AND/OR graphs, namely, Entangled, Branched and Iteratively deepened AND/OR graphs to address various HRC and TAMP problems.
We will demonstrate how AND/OR graphs are computationally fast and easily comprehensible even by a novice user. In particular AND/OR graphs are graphical representation of a whole process, mapped into a tree like graph. Analogously to a process, that is composed of starting point(s), paths or procedures and objective point, AND/OR graphs exploit the same structure. An AND/OR graph is a directed and structured graph, composed of nodes, that represent a state of a process, and hyper-arcs that represent the transition procedure from one state to another.
In Figure 2.1, a general and simple AND/OR graph is illustrated. This graph has three leaf nodes filled with blue, namely states A,B and C. The root node, that is a singleton node, filled with red and three intermediate states, D, E and F, connect leaf nodes to root node through two alternative paths, namely p1 and p2.

An AND/OR graph allows for representing *procedures* to follow, which can be decomposed in sub problems as parts of the graph, as well as the logic *relationships* among them, i.e., the graph inter connectivity. The root node conventionally represents the goal state of the process being modelled, and achieving the goal means traversing the graph from leaf

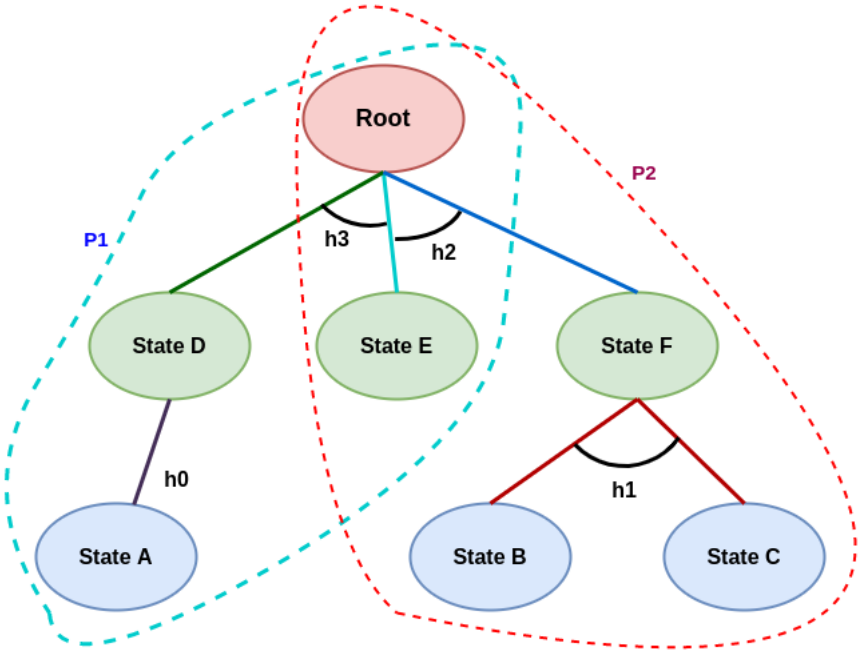


Figure 2.1 A simple AND/OR graph with three leaf nodes, three intermediate nodes and single root node

nodes to the root node via intermediate nodes and hyper-arcs according to its structure. Let us call simple AND/OR graphs as 1-layer AND/OR graphs, later in chapter 4, we will extend 1-layer AND/OR graphs and introduce n-layer and c-layer graphs as well. A 1-layer AND/OR graph G can be formally defined as a 2-ple $\langle N, H \rangle$ where N is a set of $|N|$ nodes, and H is a set of $|H|$ hyper-arcs. An hyper-arc $h \in H$ induces the set $N_c(h) \subset N$ of its *child* nodes, and the singleton $N_p(h) \subset N$ made up of a *parent* node, such that

$$h : N_c(h) \rightarrow N_p(h). \tag{2.1}$$

Furthermore, we define $n \in N$ as a *leaf* node if n is not a parent node for any hyper-arc, i.e., if $h \in H$ does not exist such that $n \in N_p(h)$, or as a *root* node if it is the only node that is not a child node for any hyper-arc, i.e., if $h \in H$ does not exist such that $n \in N_c(h)$.

In a HRC or TAMP problem, each node $n \in N$ represents a cooperation *state*, e.g., *faulty object inside box*, whereas each hyper-arc $h \in H$ represents a (possibly) *many-to-one* transition among states, i.e., activities performed by human operators and/or robots, which make the cooperation move forward, such as *the robot puts the faulty object into the box*. The relation among child nodes in hyper-arcs is the logical *and*, whereas the relation between different hyper-arcs inducing on the same parent node is the logical *or*, i.e., different hyper-

arcs inducing on the same parent node represent alternative ways for a cooperation process to move on.

Recalling Fig 2.1, in order to reach to state F, both child nodes of $h1$, meaning B and C nodes need to be feasible. This illustrates the AND logic among child nodes. In the same graph, to reach to root node, one of $h2$ or $h3$ need to be met not both of them, this illustrates the OR logic among hyper-arcs.

Each hyper-arc $h \in H$ implements the transition in (2.1) by checking the *requirements* defined by nodes in $N_c(h)$, executing *actions* associated with h, and generating *effects* compatible with the parent node. Each hyper-arc $h \in H$ executes an ordered set $A(h)$ of *actions*, such that

$$A(h) = (a_1, \ldots, a_{|A|}; \preceq), \tag{2.2}$$

where the precedence operator $\preceq$ defines the pairwise expected order of action execution. The sequence can be scripted or planned online [11]. Before an hyper-arc h is executed, all actions $a \in A(h)$ are marked as *undone*, i.e., $\mathsf{done}(a) \leftarrow false$. When one action a is executed by any agent, its status changes to $\mathsf{done}(a) \leftarrow true$. An hyper-arc $h \in H$ is marked as *solved*, i.e., $\mathsf{solved}(h) \leftarrow true$ *iff* all actions $a \in A(h)$ are done in the expected order. In a similar way, nodes $n \in N$ may be associated with a (possibly ordered) set of *processes* $P(n)$, which are typically *robot* behaviours activated in a cooperation state but not leading to a state transition.

It is possible to introduce the notion of feasibility. A node $n \in N$ is *feasible*, i.e., $\mathsf{feasible}(n) \leftarrow true$, *iff* a solved hyper-arc $h \in H$ exists, for which $n \in N_p(h)$, and $\mathsf{met}(n) \leftarrow false$, i.e.,

$$\exists h \in H. \left(\mathsf{solved}(h) \cap n \in N_p(h) \cap \neg \mathsf{met}(n)\right). \tag{2.3}$$

All leaf nodes in an AND/OR graph are usually feasible at the beginning of a cooperation process, which means that the cooperation can be performed in many ways. This is better described in Algorithm 1.

An hyper-arc $h \in H$ is *feasible*, i.e., $\mathsf{feasible}(h) \leftarrow true$, *iff* for each node $n \in N_c(h)$, $\mathsf{met}(n) \leftarrow true$ and $\mathsf{solved}(h) \leftarrow false$, i.e.,

$$\forall n \in N_c(h). \left(\mathsf{met}(n) \cap \neg \mathsf{solved}(h)\right). \tag{2.4}$$

Once an hyper-arc $h_i \in H$ is solved, all other feasible hyper-arcs $h_j \in H \setminus \{h_i\}$, which share with h_i at least one child node, i.e., $N_c(h_i) \cap N_c(h_j) \neq \emptyset$, are marked as unfeasible as shown in Algorithm 2, in order to prevent the cooperation process to consider alternative ways to cooperation that have become irrelevant.

Algorithm 1: checkNodeFeasibility

Data: $n \in G(N,H)$
Result: $feasible(n) \leftarrow true\ or\ false$

```
feasible(n) ← false;
if C_h(n) = ∅ then
    feasible(n) ← true
end
for all h ∈ H s.t n ∈ N_p(h) do
    if solved(h) then
        if met(n) ← false then
            feasible(n) ← true
        end
    end
end
return feasible(n)
```

Algorithm 2: checkHyperArcFeasibility

Data: $h \in G(N,H)$
Result: $feasible(h) \leftarrow true\ or\ false$

```
feasible(h) ← false;
for all n ∈ N s.t n ∈ N_c(h) do
    if solved(h) ← false then
        if met(n) ← true then
            feasible(h) ← true
        end
    end
end
for all h_j ∈ H \ {h} do
    if N_c(h) ∩ N_c(h_j) ≠ ∅ then
        if solved(h_j) ← true then
            feasible(h) ← false
        end
    end
end
return feasible(h)
```

Given and AND/OR graph, the multi human-robot cooperation process is modelled as a *graph traversal* procedure which, starting from a set of leaf nodes, must reach the root node by selecting hyper-arcs and reaching states in one of the available *cooperation paths*, depending on the feasibility statuses of nodes and hyper-arcs. According to the graph structure, multiple cooperation paths may exist, meaning that multiple ways to solve the task may be equally legitimate. Algorithm 3, gives a big picture of how theses paths are generated for a given AND/OR graph.

Algorithm 3: createPaths

```
Data: G(N,H)
Result: P: set of all possible cooperation paths
P ← ∅;
currentNodeSet ← ∅;
rootNode ← root(G);
currentNodeSet ← currentNodeSet ∪ rootNode;
while true do
    if C_h(currentNodeSet) = ∅ then
        break;
    else
        for all node ∈ CurrentNodeSet do
            currentNodeSet ← currentNodeSet \ node ;
            for all h ∈ C_h(node) do
                currentNodeSet ← currentNodeSet ∪ C_n(h);
                if node ∈ any    p ∈ P then
                    p.add(node);
                    p.add(h)
                else
                    p ← newPath;
                    p.add(node);
                    p.add(h);
                    P ← P ∪ p;
                end
            end
        end
    end
end
return P
```

The traversal procedure dynamically follows the cooperation path that at any time is characterised by the lowest cost. The entire algorithm has been described in [22, 20]. The

traversal procedure suggests to human operators agents actions in the hyper-arcs that are part of the path, and sends to robots the actions they must execute. Human operators can override the suggestions at any time, executing different actions, which may cause the graph to reach a state not part of the current path. When this happens, AND/OR graph tries to progress from that state onwards [20, 21]. This mechanism enables AND/OR graph to pursue an optimal path leading to the solution, while it allows human operators to choose alternative paths. As long as the multi human-robot cooperation process unfolds, and the AND/OR graph is traversed, we refer with N_f and H_f to the sets of *currently* feasible nodes and hyper-arcs, respectively. We say that an AND/OR graph G is *solved*, i.e., $\mathsf{solved}(G) \leftarrow true$, *iff* its root node $r \in N$ is met, i.e., $\mathsf{met}(r) \leftarrow true$. Otherwise, if the condition $N_f \cup H_f = \emptyset$, i.e., there are no feasible nodes nor hyper-arcs, then the multi human-robot cooperation process fails, because there is no feasible cooperation path leading to the root node.

Up to now we discussed the structure and implementation of AND/OR graphs at offline phase. Online phase includes, solving the graph starting from leaf nodes and traversing the graph toward the root node. The traversal path is assigned at run-time, given the actual status of states and human inputs. This feature of the AND/OR graphs, makes them flexible for any cooperation process. Obviously 1-layer AND/OR graphs, suffer from many limitations such as i) concurrency of multiple agents, ii) adaptability to human decisions at run-time, and the fact that AND/OR graphs, iii) before any HRC process need to be aware of quantity and quality of the process.

Therefore, in this dissertation we try to overcome these challenges by extending 1-layer AND/OR graphs. We propose three variants of AND/OR graphs to address all the three challenges we previously mentioned. To address the problem of concurrency of agents, we propose *c-layer* AND/OR graphs, that allows a team of agents made of humans and/or robots concurrently follow their own process and where needed to coordinate and schedule their tasks according to their counterpart agents. In chapter 4, we thoroughly discuss c-layer graphs and it's underlying mechanism, and implement it in a system architecture and finally validate it in a cooperative scenario.

To overcome the problem of adaptability of AND/OR graphs to human decisions at online phase, we propose branched AND/OR graphs. 1-layer AND/OR graphs, naturally are not able to deform their structure to human actions/decision. Indeed at implementation phase of AND/OR graphs, every action of human need to be encapsulated in a specific hyper-arc transition, and in case, human action intervention occurs in any other transition state, this causes the failure of the graph and consequently the failure of the overall process. *Branched* AND/OR graphs relax this limitation and allow human operators to intervene at any state

of graph transition and act while keeping the traversal procedure coherent. In chapter 4 we specifically describe these graphs and validate them in a cooperative scenario.
Eventually, to resolve the problem of the scalability of AND/OR graphs where the depth of graph should be unknown before it's implementation, we propose *iteratively deepened* AND/OR graphs. Let's consider a cooperative product defect inspection scenario, where a 1-layer AND/OR graph is able to implement only one product inspection process and for more products we need to know the number of products before-hand, moreover in this case, one needs to implement the graph for n known products that leads to a highly complex and deep graph. *iteratively deepened* AND/OR graphs, consider this difficulty and cover all these problems. In chapter 7, we implement this kind of graph, where robot arms need to pick a target object from a cluttered top-table with unknown number of objects on table and unknown number of required object rearrangements.

Chapter 3

HRC for object defect inspection

3.1 Introduction

Robots are increasingly adopted in industrial environments to carry out dangerous, repetitive, or stressful tasks. The introduction of robots in production lines has improved a number of key performance indicators, and has addressed a market-driven goods growing demand with quality products [30]. However, due to well-known limitations of robot perceptual, cognitive, and reasoning capabilities, certain tasks, which are difficult to model or require a higher-level of awareness because they cannot be easily modelled nor formalised, are still better handled by human operators. The introduction of collaborative robots (nowadays referred to as *cobots*) in recent years has contributed to relax those limitations, and implicitly promoted human working conditions [57]. Among the tasks typically considered stressful, *quality control* and *defects inspection* play a key role in defining the quality of a finished or semi-finished product. Currently, trained and expert personnel is tasked with establishing benchmarks and examining products quality, which require prolonged focus and continuous attention.

In many industries and workshops due to physical and chemical characteristics of products, identifying defects in inspection process jeopardizes the workers' safety. In this chapter, we argue that the collaboration between an experienced human operator and a robot may lead to higher rates in defects spotting, overall productivity, and safety [25, 62]. Product quality control (QC) plays a key role in assertion of final product quality and is usually done by training personnel, creating benchmarks for product quality and testing products to check for statistically significant variations. In many industries and workshops product inspection becomes risky and hard to identify defects due to the physical, chemical characteristics of products. State of the art robotic solution together with skilled inspectors in a human-

robot collaboration (HRC) framework provide fast, accurate, safe and repeatable way of inspection ensuring that final product is free of defects. Maneuverability, accuracy and strength of Cobots added to the human experience and skills lead to maximising product quality and minimising final cost. Human-robot collaboration (HRC) is defined as the purposeful interaction among humans and robots in a shared space, and it is aimed at a common goal. A natural collaboration requires a robot to perceive and correctly interpret the actions (as well as the intentions) of other humans or robots [2, 86]. This reinforces the safety, efficiency, and performance and lowers the psychological discomfort and stress in the workplace [25, 62]. An example of such natural and fluent collaboration is given by [20]; the robot reacts to and suggests the human on-the-fly while progressing the assembly scenario. The main goal of this chapter is to extend the human-robot collaboration model proposed

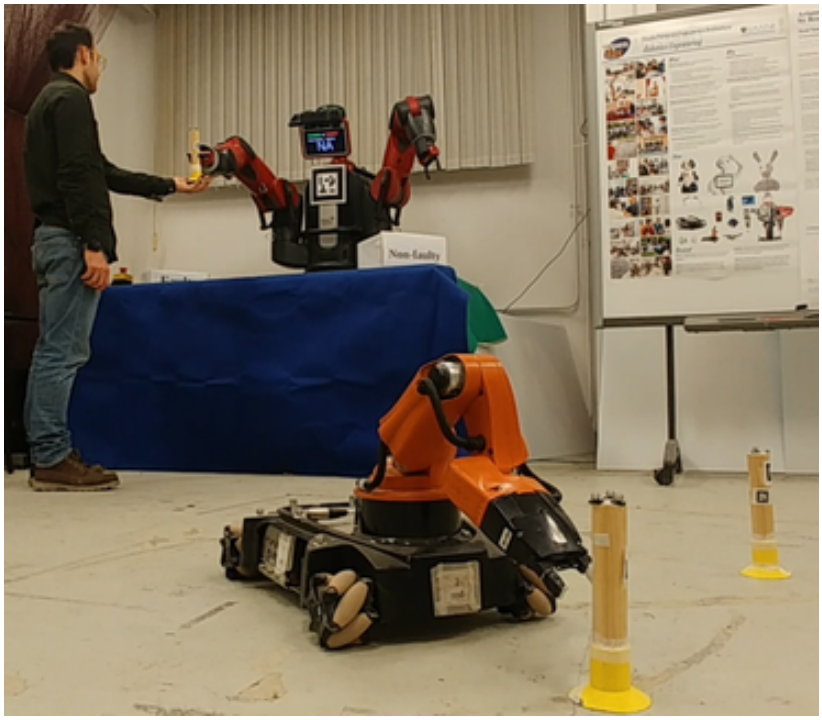


Figure 3.1 A human operator and and two robots collaborating in a product defect inspection scenario: the mobile manipulator supplies a human operator and the dual-arm manipulator with objects to inspect.

in [20], referred to as FLEXHRC, along two directions. On the one hand, to allow for a collaboration model taking multiple, heterogeneous robots into account, while the original work in [20] considered models with one human operator and one robot. On the other hand, introduce a use case whereby a human operator and a robot must collaboratively perform a defects inspection, whereas the original work focused on assembly tasks.

The scenario we consider is shown in Figure 5.1. A mobile manipulator (in our case, a Kuka youBot) picks-up objects to be inspected (wooden pieces) from a warehouse area (a marked region in the work space), and carries them out to deliver them to human operators or another robot (in our case, a dual-arm Baxter manipulator) for inspection [55]. When the object to inspect is delivered to human operators, these undertake the foreman task [47, 39], and then the object is passed to the manipulator for a further vision-based inspection. Afterwards, the manipulator sorts the object out as *faulty* or *non faulty* in two different boxes. Scenarios modelling defects inspection impose functional requirements which are partially in overlap with the ones considered in [20] for the assembly of semi-finished products.

The main functional requirement in quality control is the *validation of products quality* with a reliable estimation. In an HRC process, such a requirement can be met by a double-check carried out by an expert operator in case the defects classification accuracy as provided by the robot is below a pre-specified threshold. However, differently from the work in [56, 72, 14], whereby a visual inspection is carried out by a robot, in order to validate the quality of products an integration of auditory, tactile, and visual perception is likely to be needed [84, 36]. Such an integration is still an open issue and it is not considered in this paper.

This chapter introduces and discusses CONCHRC, a framework extending FLEXHRC that addresses the need for concurrent, multi human-robot collaboration in industrial environments, and validates the models in an inspection use case. The novelty of the approach is two-fold: (i) the design and development of an AND/OR graph based multi human-robot collaboration model that allows for concurrent, modelled, operations in a team made up of multiple human operators and/or robots; (ii) the description of a particular instance of such a cooperation model, implemented within an existing human-robot collaboration architecture, and extending it whereby a human operator, a mobile manipulator, and a dual-arm manipulator collaborate for a defect inspection purpose. In the paper, the focus is on the concurrent HRC model for the quality control task, and therefore we decided to simplify the robot perception system.

3.2 Related work

For a natural human-robot collaboration, different aspects such as safety, robot perception, task representation, and action execution must be considered when designing a collaborative-friendly workspace [41, 62]. This paper focuses on task representation when multiple human operators and/or robots group as a team to reach a common goal, which is *a priori* known to

all collaborators, either humans or robots. The uncertainties in perception, task representation and reasoning that a robot must face increase when collaborating with humans, because a natural cooperation, i.e., a one in a way similar to human-human teams [68], may require the robot to *make sense* of or even anticipate human intentions. The need arises to provide robots with reasoning capabilities about the state of the human-robot cooperation process, suitable to be executed online.

Although approaches based on offline planning and task allocation fulfil a requirement related to the effectiveness of the collaboration [48, 64], they neither ensure such a natural collaboration nor address its intrinsic uncertainties. Differently, the approaches described in [43, 65, 22, 20] are aimed at enhancing the naturalness and the flexibility of the collaboration based on online task allocation and/or contingency plans, such that the robot is able to adapt to human decisions on the spot and uncertainties. Such flexibility requires a rich perception for recognising human actions as well as the collaboration state [20].

Some of the methods applied for robot action planning in collaboration scenarios include *Markov Decision Processes* [15, 16], *Task Networks* [65, 64], *AND/OR graphs* [97, 48, 22], and STRIPS-based *planners* [11]. Among these methods, finding the priors and the reward function for Markov Decision Processes and the exponential growth of the computational load of STRIPS-based planners make them very difficult to be adopted in practice. Task Networks and AND/OR graphs ensure that the generated collaboration models are in accordance with domain expert desiderata, hence guaranteeing shared *mental* models between human operators and robots. In order to allocate tasks to human operators or robots, and to meet such collaboration constraints as limited resources, a common approach in the literature is to maximise the overall utility value of the collaboration [94] on the basis of multi-objective optimisation criteria. However, in these examples the number of human operators or robots is limited.

In order to enhance the efficiency of the collaboration, and to face the inherent limitations owing to workspace constraints, human skills, and robot capabilities, an approach can be to raise the number of human operators or heterogeneous robots involved in the collaboration. To this aim, human operators and robots must schedule their actions according to resources, timings, and skill constraints. An example can be found in [93] whereby concurrent cooperation models are formalised according to relational activity processes. The authors in that study model the cooperation and predict future actions using a Monte Carlo method along with learning by demonstration. A similar approach is adopted in [83], whereby a temporal graph plan with the consideration of action duration's has been applied. Another illustration

of concurrent HRC, with a probabilistic formulation due to uncertainties, is presented in [96], where a concurrent Markov Decision Process is adopted.

In previous work [22, 20] where we demonstrated a flexible collaboration between human operators and robots, this paper extends the notion of AND/OR graph to a concurrent model, and adopts it to model multi human-robot collaboration scenarios.

Chapter 4

Concurrent model for multi-agent cooperation

4.1 Concurrent AND/OR graphs

In this chapter, we describe an extended version of 1-layer graphs, namely n-layer AND/OR graph, and finally a concurrent model based on a constrained n-layer configuration, which we refer to as a c-layer AND/OR graph.

4.1.1 n-layer AND/OR graphs

In previous chapter, we introduced and formalised 1-layer AND/OR graphs and discussed that these graphs are limited and are not able to model cooperative processes for multiple agents concurrently. To this end we introduce *c*-layer graphs to tackle this problem. Before we present *c*-layer graphs, we formalise *n*-layer graphs that are extension of 1-layer graphs. A *n*-layer AND/OR graph G^n can be recursively defined as a 2-ple $\langle \Gamma, \Theta \rangle$ where Γ is an ordered set of $|\Gamma|$ *up to* $(n-1)$-layer AND/OR graphs, such that:

$$\Gamma = \left(G_1, \ldots, G_{|\Gamma|}; \preceq\right), \tag{4.1}$$

and Θ is a set of $|\Theta|$ pairwise transitions between them. In (5.4), the AND/OR graphs are ordered according to their layer. Lower-layer AND/OR graphs are characterised by a decreasing level of abstraction, i.e., they are aimed at modelling the HRC process more accurately. Transitions in Θ define how different AND/OR graphs in Γ are connected, and in particular model the relationship between graphs belonging to different layers.

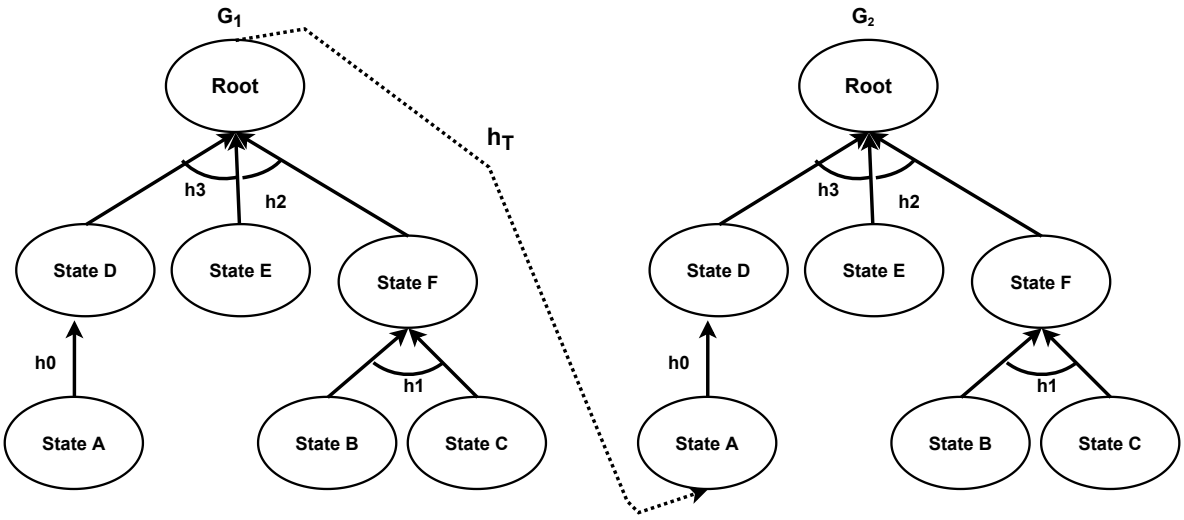


Figure 4.1 A simple n-layer AND/OR graph with two 1-layer graphs and a transition among them from G_1 to G_2 through h_T

If we recall (2.1) and we contextualise it for an AND/OR graph $G^n = \langle N^n, H^n \rangle$, we observe that a given hyper-arc in H^n represents a mapping between the set of its child nodes and the singleton parent node. We can think of a generalised version of such a mapping to encompass a whole AND/OR graph $G^{n-1} = \langle N^{n-1}, H^{n-1} \rangle$, where the set of child nodes is constituted by the set N_L^{n-1} of leaf nodes, and the singleton parent node by the graph's root node $r^{n-1} \in N^{n-1}$. As a consequence, a transition $T \in \Theta$ can be defined between a hyper-arc $h \in H^n$ and an entire AND/OR graph G^{n-1}, such that

$$T : h \rightarrow G^{n-1}, \tag{4.2}$$

subject to the fact that appropriate mappings can be defined between the set of child nodes of h and the set of leaf nodes of the deeper graph, i.e.,

$$M_1 : N_c(h) \rightarrow N_L \in N^{n-1}, \tag{4.3}$$

and between the singleton set of parent nodes of h^n and the root node of the deeper graph, i.e.,

$$M_2 : N_p(h) \rightarrow r^{n-1} \in N^{n-1}. \tag{4.4}$$

Mappings M_1 and M_2 must be such that the corresponding information in different layers should be *semantically equivalent*, i.e., it should represent the same information with a different representation granularity. The same applies for $N_p(h)$ and the root of G^{n-1}. Once

these mappings are defined, it easy to see that G^n has a tree-like structure, where graphs in Γ are nodes and transitions in Θ are edges.

An AND/OR graph G^n is feasible, i.e., $\mathsf{feasible}(G^n) \leftarrow true$ *iff* it has at least one feasible node or hyper-arc. If a transition $T \in \Theta$ exists in the form (4.2), a hyper-arc $h \in H^n$ is feasible *iff* the associated AND/OR graph G^{n-1}, is feasible, i.e.,

$$\forall T. \left(\mathsf{feasible}(h) \leftrightarrow \mathsf{feasible}(G^{n-1})\right). \tag{4.5}$$

As a consequence, when the nodes in N_L^{n-1} of G^{n-1} becomes feasible, the hyper-arc h in G^n becomes feasible as well. Furthermore, the hyper-arc h is solved *iff* the associated AND/OR graph G^{n-1} is solved, i.e.,

$$\forall T. \left(\mathsf{solved}(h) \leftrightarrow \mathsf{solved}(G^{n-1})\right). \tag{4.6}$$

Figure 4.1, illustrates a simple n-layer AND/OR graph with 2 graphs, namely G_1 and G_2. These two 1-layer AND/OR graphs are connected through h_T, as a hyper-arc for transiting from one graph to another.

4.1.2 c-layer AND/OR graphs

A concurrent AND/OR graph is modelled as a restriction of a n-layer AND/OR graph whereby the n-th layer is aimed at modelling the termination condition for the whole hierarchy of $(n-1)$-layer graphs, and the latter model different, concurrent activities part of the HRC process.

A c-layer AND/OR graph must also specify if and how nodes belonging to separate lower-layer graphs are synchronised.

Analogously to an n-layer graph, a c-layer AND/OR graph G^c can be defined as a 2-ple $\langle \Gamma^c, \Theta^c \rangle$ where Γ^c is an ordered set of $|\Gamma^c|$ *up to* $(n-1)$-layer AND/OR graphs, such that:

$$\Gamma^c = \left(G_1, \ldots, G_{|\Gamma^c|}; \preceq\right), \tag{4.7}$$

and Θ^c is a set of $|\Theta^c|$ pairwise transitions between them.

Whilst the considerations related to n-layer AND/OR graphs apply for c-layer AND/OR graphs, the composition of the constituting sets of nodes and hyper-arcs may differ. Let us recall that for a generic AND/OR graph G we refer to N as its set of nodes, and with H as its set of hyper-arcs, and let us consider two AND/OR graphs G_i and $G_j \in \Gamma^c$. Let us

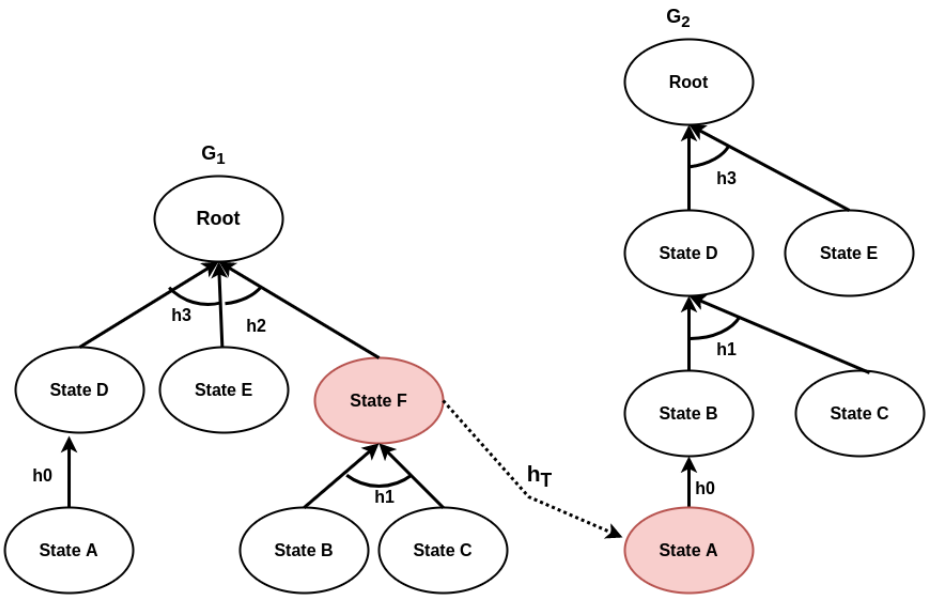


Figure 4.2 a simple c-layer AND/OR graph with two 1-layer AND/OR graphs that depend on each other through h_T, Graph G_2 starts when node F of graph G_1 is met.

limit ourselves to a weak notion of independence between graphs. We consider G_i and G_j as mutually independent *iff* there is no node in G_i (respectively, G_j) that needs to be met before another node of G_j (respectively, G_i). If this is the case, G_i (respectively, G_j) can be modelled as a generic n-layer AND/OR graphs $\langle N_i, H_i \rangle$ (respectively, $\langle N_j, H_j \rangle$). Otherwise, if G_i is dependent on G_j, i.e., a node n_j in G_j must be met before another node n_i in G_i can be met, we need to formally model it as an external dependence.

To this aim, and in general terms, we augment the set of nodes G_i with a set of dependence nodes, whose associated logic predicates met are entangled with the corresponding nodes in G_j, such that their truth values always correspond. A node n^e of an AND/OR graph G_i is said to be *entangled* with a node n_j of an AND/OR graph G_j, with $i \neq j$, *iff* for that node

$$\mathsf{met}(n^e) \leftrightarrow \mathsf{met}(n_j) \tag{4.8}$$

and n^e is a leaf node for G_i, i.e., $N_c(n^e) = \emptyset$. Then, a *dependent* AND/OR graph G_i is defined as a 2-ple $\langle N_i^c, H_i^c \rangle$, such that $N_i^c = N_i \cup \{n_1^e, \ldots, n_\eta^e\}$, i.e., the union between the set of nodes N_i as if the graph were not dependent on any other graph, plus the set of the entangled nodes, and $H_i^c = H_i \cup \{h_1^e, \ldots, h_\lambda^e\}$, i.e., the union between the set of hyper-arcs H_i as if the graph were not dependent on any other graph, plus the set of the hyper-arcs reliant on entangled nodes.

Figure 4.1.2 depicts a simple c-layer AND/OR graphs, with two simple 1-layer AND/OR graphs that depend on each other through h_T, meaning that, graph G_2, can start it's procedure iff node F of graph G_1 is met. Indeed node F in graph G_1 and node A in graph G_2 are entangled nodes.

4.2 Systems architecture

Figure 5.3 depicts the overall architecture of the CONCHRC framework. The architecture is made up of three layers, including a *perception* layer in green, a *representation* layer in blue, and an *action* layer in red. The perception layer provides information regarding the activities carried out by human operators, a part's defect status, and object locations in the robot workspace. The representation layer forms the concurrency model, stores the necessary knowledge, and manages task execution to reach the collaboration goal. The action level simulates and executes robot actions.

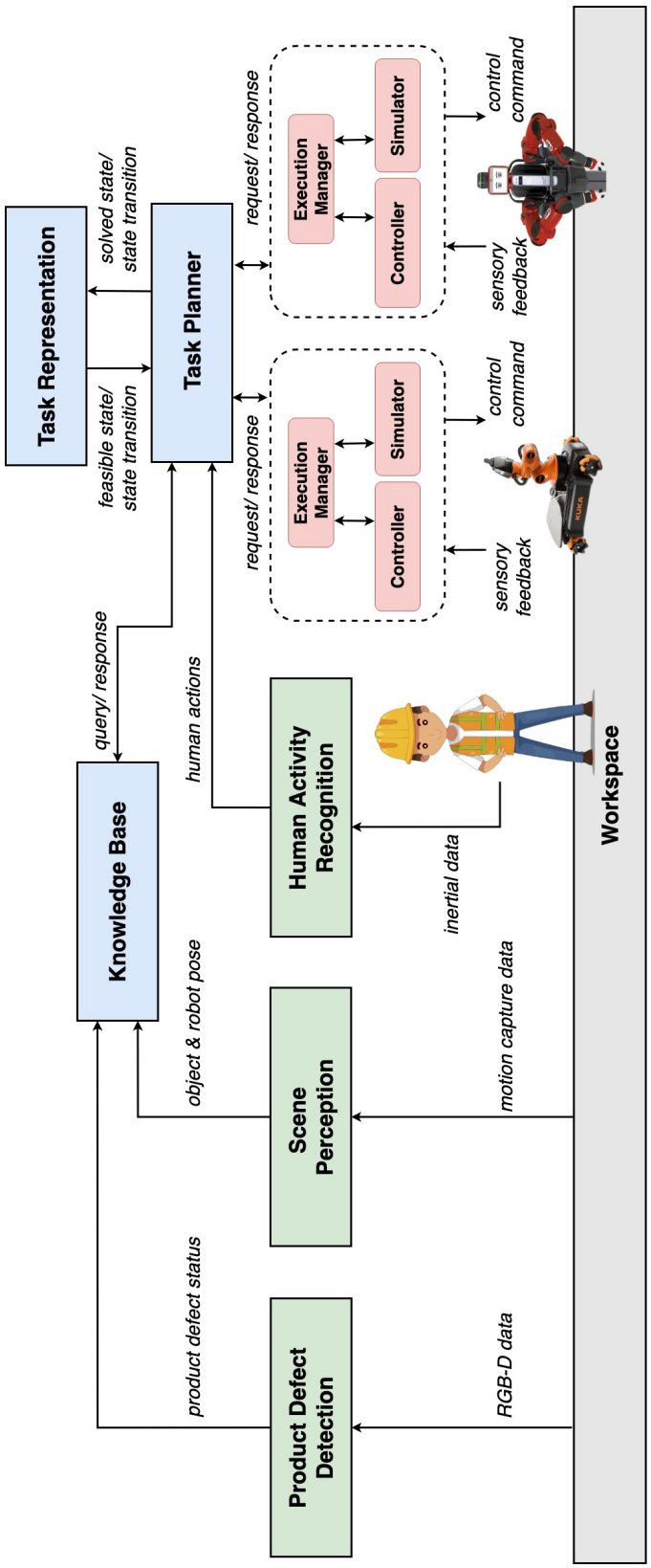


Figure 4.3 System's architecture for a multi human-robot collaboration model in a defects detection scenario.

The perception layer encapsulates three modules, which are called *Human Activity Recognition*, *Product Defect Detection*, and *Scene Perception*. The latter two modules provide the *Knowledge Base* module with information about the status of the workspace, human operators, and robots, whereas the former communicates detected human activities to the *Task Planner*. *Human Activity Recogntion* obtains inertial data originating from wearable sensors worn by human operators, and run a series of algorithms to detect and classify performed actions.
Those are modelled using *Gaussian Mixture Modelling* (GMM) and *Regression* [7, 22]. In our setup, defects detection is considered as a classification problem. *Product Defect Detection* exploits the images coming from a robot-centric camera to detect defects.

The action layer is made up of three modules, namely *Robot Execution Manager*, *Simulator*, and *Controller*. The *Robot Execution Manager* module receives discrete, symbolic commands from the *Task Planner*, maps them to actual values, and drives the behaviour of the *Controller* or the *Simulator*. This module retrieves information about the workspace, human operators and robots from the *Knowledge Base*.
The *Robot Execution Manager* is in charge of sequencing robot behaviours, on the basis of the plan as provided by the *Task Representation* module. It also provides an acknowledgement to the *Task Planner* upon the execution of a command by the robots.
The *Simulator* module is aimed at predicting the outcome of robot behaviours before their actual execution. It simulates a closed-loop model of the robot and the controller, by solving the ordinary differential equations online. The *Controller* receives the configuration (in joint space) or the task space command (in the Cartesian space) from the *Robot Execution Manager*. It computes the joints velocity reference values at each control time step to the robot, while receiving feedback from it [82].

The representation layer embeds *Task Representation*, *Task Planner*, and the *Knowledge Base* module. In the CONCHRC, an AND/OR graph with several layers represents the collaborative task [22]. In order to model concurrency in a multi-agent collaboration scenario, the AND/OR graph based FLEXHRC framework has been extended, as described in the next Section. Along with the AND/OR graph, the *Task Planner* module is in charge of decision making and the adaptation of the ongoing parallel tasks. To do so, the *Task Planner* provides a set of achieved cooperation states or transitions between states to the *Task Representation* module, and receives the set of allowed cooperation states and transitions with the associated costs to follow.
Later, it associates each state or state transition with an ordered set of actions, and according to the workspace's, human operator's, and robot's status, along with online simulation results,

it assigns actions to the either human operators or robots. Finally, it informs each human operator or robot involved in the cooperation about the action to follow.
Once an action is carried out, it receives the acknowledgement from the action level and updates its internal structure. The *Knowledge Base* stores all relevant information to make the cooperation progress, as better described in [22].

4.2.1 Task planner

Task planner is the AND/OR graph solver and is in connection with action level, knowledge base and human activity recognition modules. As we previously mentioned AND/OR graph embeds nodes as states of cooperation and hyper-arcs as transition rule among the states.
However AND/OR graph is not aware of the transition contents, that are a set of ordered actions done by robot or human operators. Task planner queries AND/OR graph for next feasible states and hyper-arcs and manages state transition exploiting it's internal algorithm. Once task planner receives set of feasible states and hyper-arcs, it finds optimal state transition of the received set and decomposes the optimal state transition into a set of action and agent table. please note that all the information for state transition, i.e., actions, their order and their corresponding responsible agents are loaded to task planner at it's offline phase. Once task planner creates action agent table, it translates these actions from high-level commands, e.g., *approach object-pregrasp Right-Arm* to low-level commands of motion. This is mainly done by sending high level commands to Execution Manger that is responsible mainly for inter-mediating between high and low levels of abstractions.
Execution manager is aware of the action templates, and easily interprets the messages calling knowledge base module. It sends a request to knowledge base module asking for *object-pregrasp* pose for instance. This pose is then sent back to Execution Manager. A better description of Task Planner is shown in algorithms 4,5 and 6;

4.2.2 knowledge base

Knowledge base acts as a database in the overall architecture. It is responsible for storing data coming from perception layer and responding to any request from other modules. In particular, it is actively updating it's internal data such as, position of robots and objects in the scene, status of objects, position and orientation of robot arms to approach and grasp objects. It's noteworthy to mention that knowledge base at the beginning, reads a file of primary data to start with. A short description of knowledge base module is drawn in algorithm 7.

Algorithm 4: Task Planner

Data: AND/OR graph, G(N, H)
Result: Solved AND/OR graph

```
1  weights ← ∅;
2  while true do
3      H_f, N_f = RequestFeasibleStates;
4      if H_f = ∅ then
5          return Failure
6      end
7      for h ∈ H_f do
8          h_o = findOptimalState
9      end
10     actions, agents = decompose(h_0);
11     for agent ∈ agents do
12         for action ∈ actions do
13             doable, weight = checkIfDoable(action, agent) ;      /* Execution
                Manager */
14             if doable then
15                 weights ← weights ∪ weight;
16             else
17                 break;
18             end
19         end
20     end
21     if weight = ∅ then
22         H_f ← H_f \ h_o;
23         Go to line 4 ;
24     end
25     feasibleAgent, actionsSet = findMinimumCost(weights);
26     for action ∈ actionsSet do
27         actionDone ← executeAcion(action, feasibleAgent) ;      /* Execution
            Manager */
28         if actionDone = false then
29             H_f ← H_f \ h_o;
30             Go to line 4 ;
31         end
32     end
33     setSolvedState(h_0);
34 end
```

Algorithm 5: checkIfDoable [Execution Manager]

Data: agent, action
Result: if assigned agent is able to execute action, if so, calculate action weight

```
actionType, actionParameter ← split(action);
funcion ← selectActionFunction(actionType);
numercalValue ← retriveValue(actionParemeter) ;      /* Knowledge Base */
doable, weights ← function.simulate(agent, numericalValue) ; /* simulator */
return doable, weights
```

Algorithm 6: executeAction [Execution Manager]

Data: feasibleAgent, action
Result: action done or not

```
actionType, actionParameter ← split(action);
funcion ← selectActionFunction(actionType);
numercalValue ← retriveValue(actionParemeter) ;      /* Knowledge Base */
done ← function.execute(feasibleAgent, numericalValue) ;  /* Controller */
return done
```

Algorithm 7: Knowledge Base

Data: perception layer
Result: updated data

```
dataTable ← readDataFile();
while true do
    robotPos ← receiveRobotPos();
    dataTable ← updateDataTable(robotPos);
    objectsPos ← receiveObjectsPos();
    dataTable ← updateDataTable(objectsPos);
    if requestUpdateDefectStatus then
        defectStatus ← requestUpdateDefectStatus;
        dataTable ← updateDataTable(defectStatus);
    end
    if reciveQuery then
        retriveDataandRespond();
    end
end
```

4.3 Experimental Validation

4.3.1 Implementation of the Multi Human-Robot Collaboration Process for Defects Inspection

In order to validate the effectiveness of CONCHRC, we implemented an abstract defects inspection scenario. The scenario has been briefly described in the Introduction, and is represented in Figure 5.1.

A Kuka youBot omni-directional mobile manipulator is used to pick up objects from a *warehouse area*, and brings them close to the *defects inspection* cell, where a human operator and a dual-arm Baxter robot are expected to collaborate. The youBot and the objects to be manipulated are localised in the work space using an external motion capture system based on passive markers, i.e., a system composed of 8 OptiTrack-Flex 13 motion capture cameras. Baxter is provided with the standard grippers, and is equipped also with a RGB-D camera mounted on the robot *head* and pointing downward, which is used to acquire images for defects inspection. Since, in our case, the focus is on the multi human-robot collaboration process, we decided to over-simplify the inspection, which is surrogated using QR tags corresponding to *faulty*, *non faulty*, *Na*, respectively. Actions carried out by human operators are perceived via their inertial blueprint via an LG G Watch R (W110) smartwatch, worn at the right wrist. Data are transmitted through a standard WiFi link to a workstation. The workstation is equipped with an Intel(R) core i7-8700 @ 3.2 GHz $\times$ 12 CPUs and 16 GB of RAM. The architecture is developed using C++ and Python under ROS Kinetic.

There are upper bounds to the maximum angular velocity of arm joints for both the Baxter and the youBot, i.e., 0.6 rad/s. Limits on the youBot's linear and angular velocities are 0.4 m/s and 0.3 rad/s, respectively. These limits are applied to both simulated and real robots. Action models foreseen for human activity recognition are simply *pick up* and *put down*. Instead, actions used for Baxter arms include *approach*, *grasp*, *ungrasp*, *hold on*, *stop*, *check object status*, whereas for the youBot arm we considered only *approach*.

Our scenario includes three physical agents, i.e., a human operator, Baxter and youBot, but five *logical* agents, i.e., the operator, the Baxter left arm, the Baxter right arm, the youBot base, and the youBot arm. However, one planner manages both Baxter arms, and likewise one planner manages the youBot base and arm, so they are used sequentially. In the scenario, objects are randomly placed in the warehouse area.

Objects are cylinders labeled with three different QR code types (Figure 4.5).

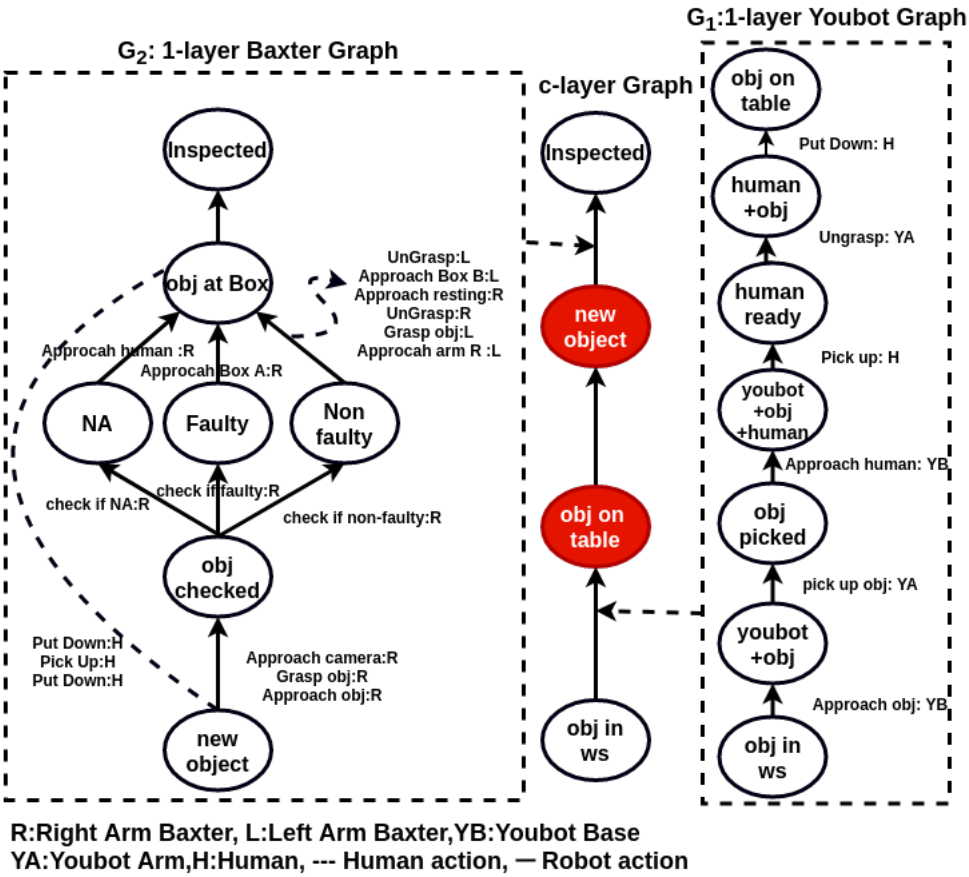


Figure 4.4 The collaboration graph for defects inspection.

The youBot must find each object, move towards it, pick it, take it to the area where the human operator and the Baxter are located, and hand it over the operator. This sequence is repeated until all objects are delivered.

On the other side of the collaboration scenario, the Baxter starts its operations when the human operator puts down an object on the table in front of the robot. By default, its right arm is used to pick the object up, and to check whether it is faulty, non-faulty or the defect cannot be assessed.

If the object is faulty, it is placed in a *faulty* box close to the right arm, or in case of a non-faulty object, the object is handed over to the left arm to be placed in a *non-faulty* box. If the object level of defects cannot be assessed, then it is handed back to the human operator for an *ad hoc* assessment. This process is repeated for all objects.

Figure 4.5 Four tagged cylinders used in our scenario.

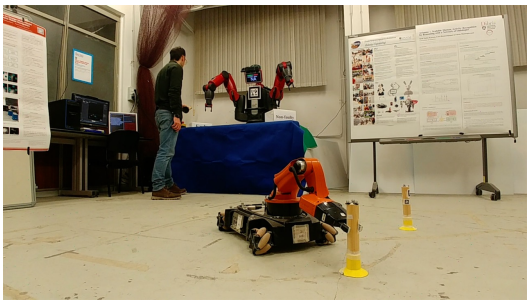
4.3.2 Description of the Experiment

Figure 4.4 shows a c-layer concurrent AND/OR graph, which is composed of two 1-layer AND/OR graphs, for the youBot (G_1) and the Baxter (G_2), respectively. Entangled nodes of both graphs are depicted in red, which makes graph G_2 dependent on graph G_1. In order for the leaf node of G_2 (i.e., *new object*) to be feasible, the root node of G_1 (i.e., *obj on table*) must be met.

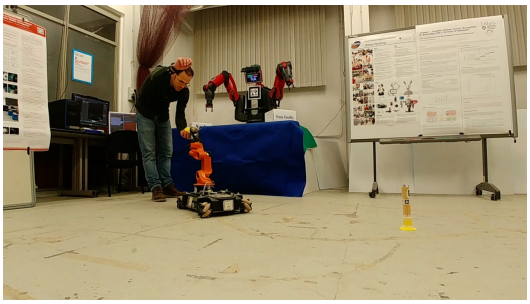
During the HRC process, the human operator is typically close to the Baxter, as shown in Figure 5.6. When the youBot approaches, the operator executes a *discrete gesture* moving an arm upward in order to announce a *pick up* action. Once the gesture is detected, youBot releases the object opening the end-effector to hand it over. Afterwards, the operator announces via a *put down* gesture the fact that the object to inspect has been located on the table for the Baxter to start inspection.

Figure 5.6 shows a typical run of the collaboration process. In the initial configuration, shown in Figure 5.6a, both the human operator and the robots are in stand by mode. The youBot moves towards the next object to inspect (*obj in ws* state), according to graph G_1. The object is selected on the basis of the time it takes to perform the whole operation in simulation. After approaching the object (*youbot+obj*), the youBot's arm attempts grasping (Figure 5.6b), and then picking it up (*obj picked*). In the meantime, the Baxter is waiting for human operator actions to start collaboration. The youBot moves towards the human

(a)



(b)



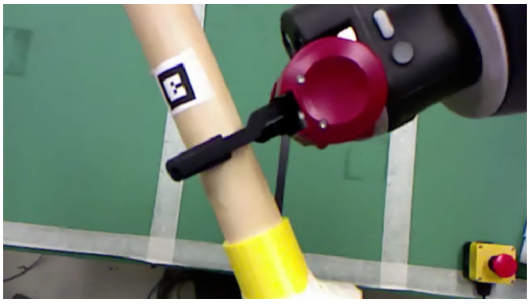
(c)


Faulty
Non-faulty

(d)


Faulty
Non-faulty

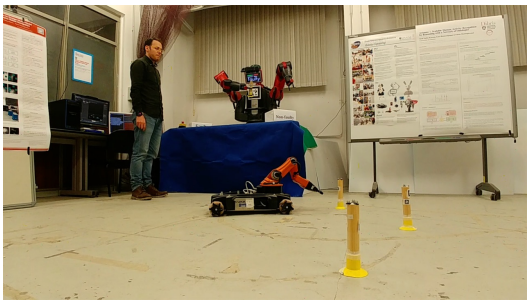
(e)



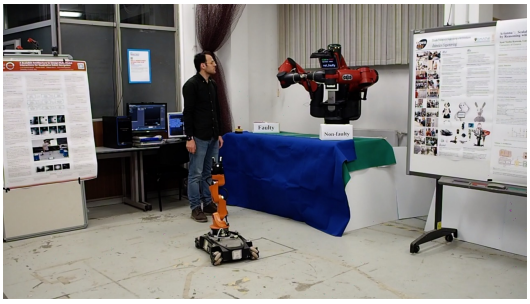
(f)


Faulty
Non-faulty

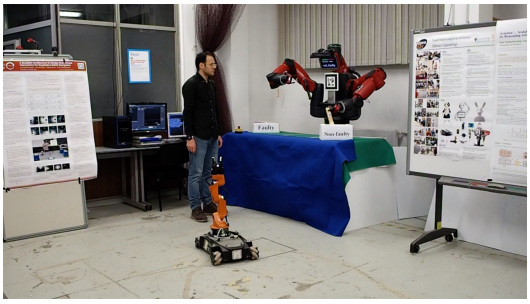
(g)



(h)


Faulty
Non-faulty

(i)


Faulty
Non-faulty

(j)

(k)

(l)

Figure 4.6 Different stages of the Human-Robot Collaboration in which the operator alters the execution of the plan online.

operator (Figure 5.6c), and waits for a command to release the object (*youbot+obj+human* state). This is done by the operator by moving an arm upward (*human ready*), which implies the youBot to open the gripper.

The operator takes the object (*human+obj*) and puts it down (*obj on table*) on the table. The operator, then, can keep moving downward one arm, therefore notifying to the Baxter that an object is on the table (Figure 5.6d).

An entangled node (*new object*) becomes feasible after the root node of G_1 is met. It is noteworthy that in some cases the youBot was not able to grasp objects properly, or dropped it actually before handover could occur. Furthermore, it happened that human actions were not recognised, which required the operator to repeat them. In these cases it is the operator's responsibility to handle the situation by taking appropriate actions in order to make the collaboration fluent. Upon the notification of the appropriate operator gesture, the Baxter starts grasping the object (Figure 5.6e) and moves it in order to place it in front of the head-mounted camera, rotating it (*obj checked*) for defects inspection (Figure 5.6f).

In Figure 5.6g, it is shown how the object is recognised as *faulty*, and therefore the right arm places it in the *faulty box* (*obj at box*). While the Baxter is inspecting the object, the youBot continues to look for other objects (Figure 5.6h). After a while, as shown in Figure 5.6i, one of the objects is classified as *non faulty*. Since the related box cannot be reached by the Baxter right arm, an handover in-between the two arms is executed (Figure 5.6j).

In case the assessment cannot be done (this is simulated with a specific QR tag), the graph reaches a *NA* state, which implies that the Baxter requires the human operator to inspect the object directly (Figure 5.6k).

After all objects are inspected (*inspected* state), the human operator performs a check (Figure 5.6l).

In order to perform a realistic computational assessment of the architecture, the whole sce-

Table 4.1 Baxter-related activities.

Module	Avg. time [s]	Avg. time [%]	Std. dev. [s]
Task Representation	0.52	0.21	0.01
Task Planner	0.02	0.008	0.003
Simulator	3.69	1.49	0.24
Baxter actions	203.00	82.00	5.00
Human actions	39.00	15.80	6.00
Total	246.75	100.00	11.253

nario has been tested five times. Results can be seen in Tables 4.1 and 4.2, where times are related to the whole experiments[1]. Statistics presented in Table 4.1 and Table 4.2 seem to indicate that the representation and planning modules together require less than 1% of the overall execution time, whereas the major portion of collaboration time is related to human or robot actions. The standard deviation related to task planners and the representation modules for both robots are low enough to be neglected, and imposes no latency in collaboration proccess.

Table 4.2 youBot-related activities.

Module	Avg. time [s]	Avg. time [%]	Std. dev. [s]
Task Representation	0.43	0.13	0.02
Task Planner	0.02	0.00	0.004
Simulator	2.74	0.79	0.40
youBot actions	268.00	86.00	14.00
Human actions	39.00	12.50	6.00
Total	310.19	100.00	20.424

4.3.3 Discussion

On the basis of the experiments we carried out, it is possible to make two different remarks. The first is related to the robustness associated with the overall process. In spite of such faults as unsuccessful robot grasps, or issues related to false positives or negatives when monitoring the activities carried out by human operators, the inherent flexibility of CONCHRC allows human operators to intervene and manage these issues. This is even more relevant considering

[1] A video is available at https://youtu.be/0aOOeqCL2So.

that our current setup does not focus on such a robustness level.
The second is the insight that using parallel instances of AND/OR graph representation layers seems to be more efficient with respect to an equivalent, common, single instance model. We observed that the adoption of CONCHRC reduces the overall idle time considerably.
This is an obvious consequence of the fact that the total time needed for a multi human-robot collaboration process to conclude is determined by the maximum one associated with the longest execution branch in the graph. On the contrary, if the HRC process were implemented as a single, non concurrent, model, then the total time would correspond to the sum of all times associated with single cooperation paths. As an example, in our scenario CONCHRC allows for a total collaboration time equal to 310.19 *s*, whereas an equivalent implementation using FLEXHRC the total collaboration time can be up to 866.94 *s*.

Chapter 5

Flexible and adaptable human-robot collaboration

5.1 Introduction

Despite significant results achieved in different research fields of robotics, robots are far from substituting human workers because of their limited cognitive capabilities, non-reliable perception systems, and non-adaptive decision-making processes. For these reasons, they often perform simple specialized tasks, and they have to rely on human help for more complex ones. The Industry 4.0 paradigm envisions a close relationship between robots and human operators to overcome these limitations and achieve higher shop-floor flexibility. In this context, robots should both operate autonomously and collaborate with operators.

The execution of robotic tasks needs a plan detailing all the necessary actions and their execution order. This plan, in collaborative scenarios, should be shared with an operator and, to favour an intuitive and flexible human-robot collaboration, the system should adapt it to the operator's actions and sudden variations of the task. We can formalize these ideas in two requirements:

> [R1] The system should not fix a priori the task allocation between human and robot. The operator should be free to choose its following action, whereas the robot should accommodate it;
>
> [R2] Since, in a dynamic environment, the target task can have slight alterations. The collaboration should be robust to variations in the plan or the robot's workspace. This flexibility can be achieved either using Artificial Intelligence (AI) techniques or leveraging the operator capabilities.

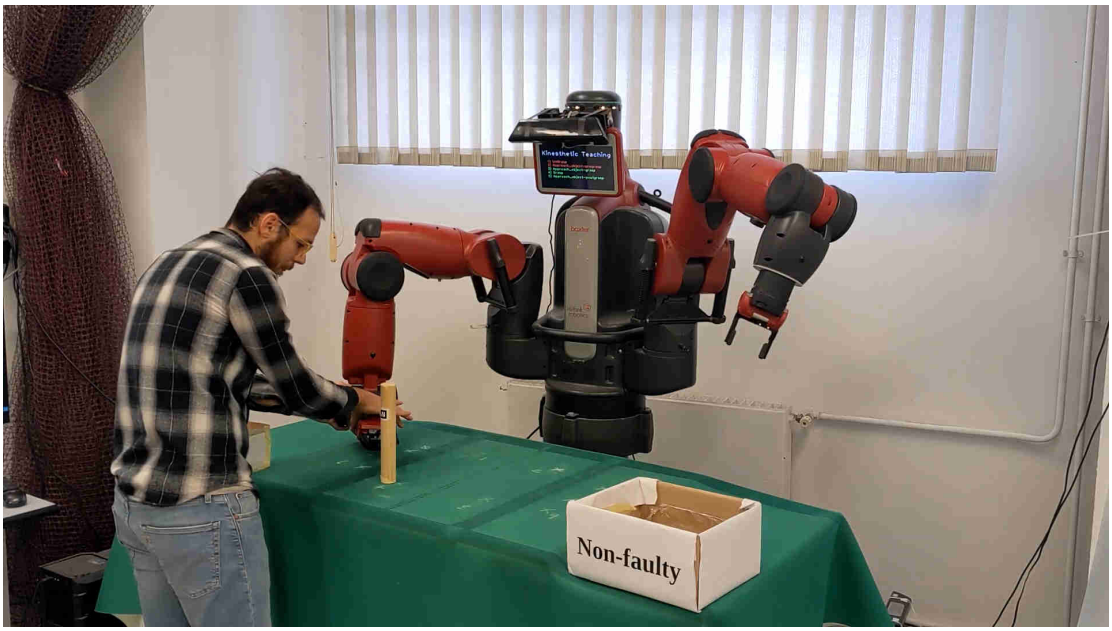


Figure 5.1 A human operator using kinesthetic teaching to update a grasping pose for a robot manipulator.

Since they fix the plan and allocate the tasks to the human and the robot offline, simple Human-Robot Collaboration (HRC) frameworks do not satisfy [R1] and [R2]. An example of this approach is Chaski, a task-level executive that, given a shared plan, enables a robot to collaboratively execute it with a human [80]. To allocate actions between the human and the robot, Chen [13] proposed evolutionary algorithms, while other studies used frameworks considering human ergonomic [94] or geometrical and physical properties of the objects involved in the task [66]. Finally, AND/OR graphs have been proposed to represent collaborative plans [48].

More advanced HRC frameworks satisfy [R1], allowing the operator to act freely. Levin [65] proposed Pike, a method using temporally flexible plans and recognition of human intent to model and execute a collaborative task in which the robot chooses online the subsequent action according to what the human is doing. Similarly, FlexHRC+ is a framework integrating AND/OR graphs in a software architecture that removes any constraint in the operator workflow [21]. Other approaches used AND/OR graphs with probabilistic graphical models to predict human actions timing [43], and Markov decision processes have been used to describe and formalize concurrent cooperation [93]. However, none of these approaches allows online plan adaptation.

At the same time, researchers explored Learning from Demonstration (LfD) to provide an easy way to program robots for non-expert users. LfD consists of extracting task information

or learning the complete plan while observing a human demonstrating it. Human demonstrations can be provided using different approaches such as teleoperation [1], perception and observation [78], or kinesthetic teaching [3]. Some studies tried to integrate the learning aspect with collaborative frameworks. Toussaint [93] demonstrated that their framework, relying on Markov decision processes, can learn new tasks using both Direct Policy Learning and Inverse Reinforcement Learning. While in a more recent work, humans can teach new tasks using vocal instructions [4]. However, in both cases, no online adaptation is possible since learning should occur before the collaboration begins.

In this chapter, we propose a framework that satisfies both the flexibility [R1] and adaptability [R2] requirements. We adopted AND/OR graphs because it has been already proved that they can handle flexible human-robot collaboration [21, 43], and we extended them creating Branched AND/OR graphs. In this extension, branches to the main graph can be generated online, allowing operators to adapt the plan execution. As a normal AND/OR graph, the designer should define the branch structure a priori according to the problem they solve.

In particular, we implemented two kinds of branches: i) a first branch giving the operator the possibility to modify the plan variables (e.g., grasping or releasing poses) using kinesthetic teaching; ii) a second branch allowing the operator to move forward, and backwards, in the plan execution. We have then deployed and tested Branched AND/OR graphs in an architecture inspired by FlexHRC [21] for a defect inspection application where the operator can invoke branches through a gestural interface.

In our testing scenario, a dual-arm robot picks an object from a table, inspects it, and decides to place it in one of two boxes according to the inspection result.

5.2 Branched AND/OR graphs

In this section, we provide a brief overview of *AND/OR Graphs*, and we introduce *Branched AND/OR Graphs*, an extension allowing online plan modifications. Before that we recap AND/OR graphs and re-formalise them to adapt better to *Branched AND/OR Graphs* definitions.

5.2.1 AND/OR graphs

AND/OR graphs (AOG) can represent a plan, divided into its sub tasks with proper logical relations. They are defined as $G = \langle N, H \rangle$ where N is the set of nodes in the graph, and H is

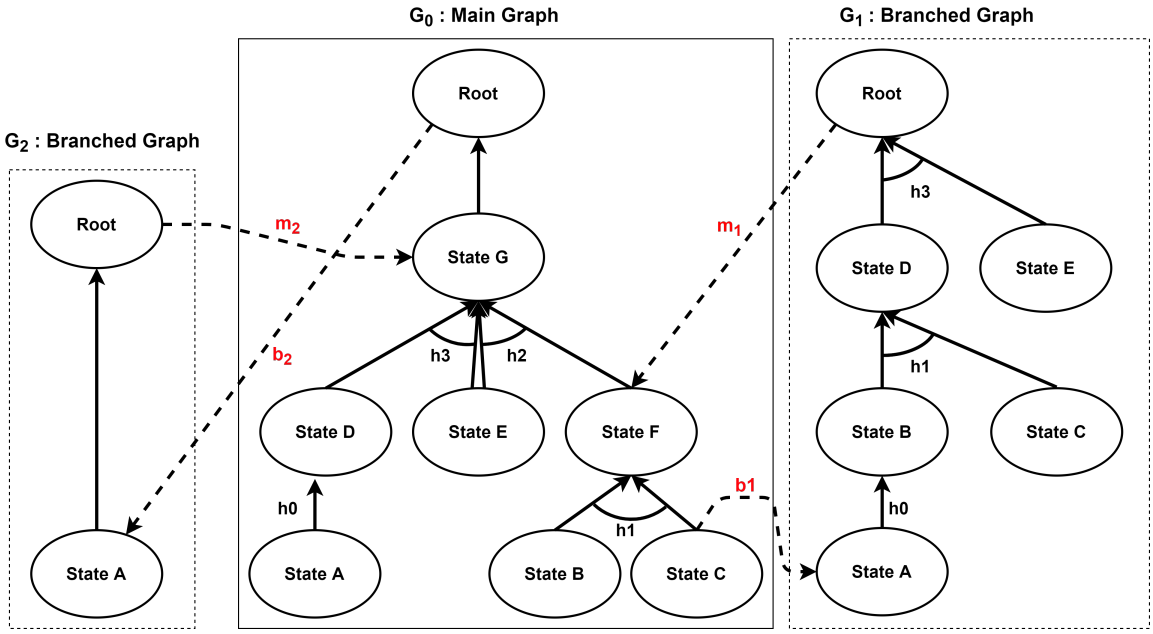


Figure 5.2 A main AND/OR graph (G_0), in the centre, and two branches (G_1 and G_2), on the left and right side.

the set of hyper-arcs. Each node ($n \in N$) represents a different plan execution state, and each hyper-arc ($h \in H$) can connect one or more child nodes ($n \in N_c(h) \subset N$) to their parent node ($n \in N_p(h) \subset N$). In the AOG formalism, $N_c(h)$ and $N_p(h)$ are respectively the set of child and parent nodes associated to an hyper-arc h. The relation, established by the hyper-arc, between child nodes and the parent node is formalized as:

$$h : N_c(h) \rightarrow N_p(h) \tag{5.1}$$

and it is equivalent to a logical AND (i.e., all the child nodes should be solved to proceed to the parent node). Notice that $|N_c(h)| \geq 1$ and $|N_p(h)| = 1$. At the same time, a parent node can be connected to multiple hyper-arcs. All the hyper-arcs inducing on the same parent node are in logical OR (i.e., only the child nodes from one hyper-arc should be solved to proceed to the parent node). The graphical representation of a hyper-arc is an arc connecting all the edges from the child to the parent node (see Figure 5.2). Considering the AOG conceptualization introduced in [21] each hyper-arc is associated with a set of actions $A(h)$ that should be executed to reach the parent node. Once all the actions associated to an hyper-arc are terminated, the hyper-arc is marked as solved ($\mathsf{solved}(h) \leftarrow \mathit{true}$). Similarly, each node is associated with a set of processes $P(n)$ whose execution does not impose state

transitions. When each process associated to a node n is executed, the node is marked as met ($\mathsf{met}(n) \leftarrow \mathit{true}$).

In AOGs, the root node defines the goal state and, to reach it, the graph should be traversed starting from the leaf node[1].

5.2.2 Branched AND/OR graphs

Given that $G_0 = \langle N_0, H_0 \rangle$ is the graph representing the plan associated with the main task, a Branched AOG is a new graph $G_i = \langle N_i, H_i \rangle$ for which we define a set of transitions $T_i = \{b_i, m_i\}$. The two transitions in T_i represent the branching b_i, that activates G_i, and the merging m_i, restoring the execution of the main graph G_0. These two transitions are simply two new hyper-arcs and we can describe them using Eq. 5.1 as:

$$\begin{aligned} b_i &: N_c(b_i) \subset N_0 \rightarrow N_p(b_i) \subset N_i \\ m_i &: N_c(m_i) \subset N_i \rightarrow N_p(m_i) \subset N_0 \end{aligned} \tag{5.2}$$

where we define the two sets of child nodes as singleton ($|N_c(b_i)| = |N_c(m_i)| = 1$). This constraint implies that hyper-arcs, connecting the branched graph with the main one, have only one child node. We refer to the node from which the main graph branches as n_{bi}, and the node where the branch merges as n_{mi}. Figure 5.2 represents an AOG subject to two branches G_1 and G_2.

To preserve a coherent AOG structure, we impose two additional conditions:

A) the node from which the main graph is branched ($n_{bi} \in N_c(b_i)$) should be met ($\mathsf{met}(n_{bi}) \leftarrow \mathit{true}$) and, all hyper-arcs for which the node $n_{bi} \in H_c(h)$ is a child should not be solved, i.e.,

$$\begin{aligned} &\mathsf{met}(n_{bi}) \leftarrow \mathit{true} \\ &\forall h \in H_0 | n_{bi} \in N_c(h), \mathsf{solved}(h) \leftarrow \mathit{false} \end{aligned} \tag{5.3}$$

B) the node n_{mi}, where the branch is merged, is marked as met ($\mathsf{met}(n_{mi}) \leftarrow \mathit{true}$). Furthermore, if n_{mi} was previously marked as met, all the hyper-arcs for which it is a child are set as unsolved, and all their parents are marked as not met ($\mathsf{met}(n) \leftarrow \mathit{false}$).

The last condition is needed to address cases in which the merging node is a node in the AOG preceding the branching node. In Figure 5.2, this case is represented by the branch G_2 for which the branching node is *root* and the merging node is *State G*.

[1] The reader may refer to [23] for a more in-depth description of AOG.

As shown in Figure 5.2, the Main AOG can be branched multiple times. However, since the branching starts always from the main graph, before a new branch can start the previous one should be merged.

Every time a new branch is performed the AOG is enlarged. This process can be described introducing the concept of AOG *networks* (AOGN). An AOGN Γ_k, where $k \in \mathbb{N}$ is recursively defined as:

$$\begin{aligned} \Gamma_k &= (G_k, \Gamma_{k-1}) \\ \Gamma_0 &= G_0 \end{aligned} \tag{5.4}$$

where G_0 is the main graph and G_k is the $k-th$ branched graph. As with normal AOG, the structure of the main graph and possible branches is designed offline. However, branches are added online to the main graph granting high flexibility to the collaboration process. The branching mechanism can be associated with automatic triggers or an external input (e.g., a command from the operator). In the following sections, we will explore how to integrate branched AOG in HRC architectures and how they can be used to alter the original plan execution.

5.3 Systems architecture

The architecture handling the Human-Robot collaboration process in our scenario is inspired by [21]. This architecture contains some blocks specialized for the defect inspection use case but, it can be easily adapted to other scenarios. In our use case, the collaboration involves the operator and the two arms of a dual-arm manipulator.

Our architecture is organized in three layers: *perception* (in green in Figure 5.3), *representation* (in blue), and *action* (in red). The perception layer collects and organizes information from all the sensors about the operator and the robot's work space, fundamental to make informed decisions. The representation layer handles the plan structure and decision making using AOGs and stores all the relevant information for plan execution (e.g., object positions). Finally, the action layer checks the feasibility of robot motions and executes them.

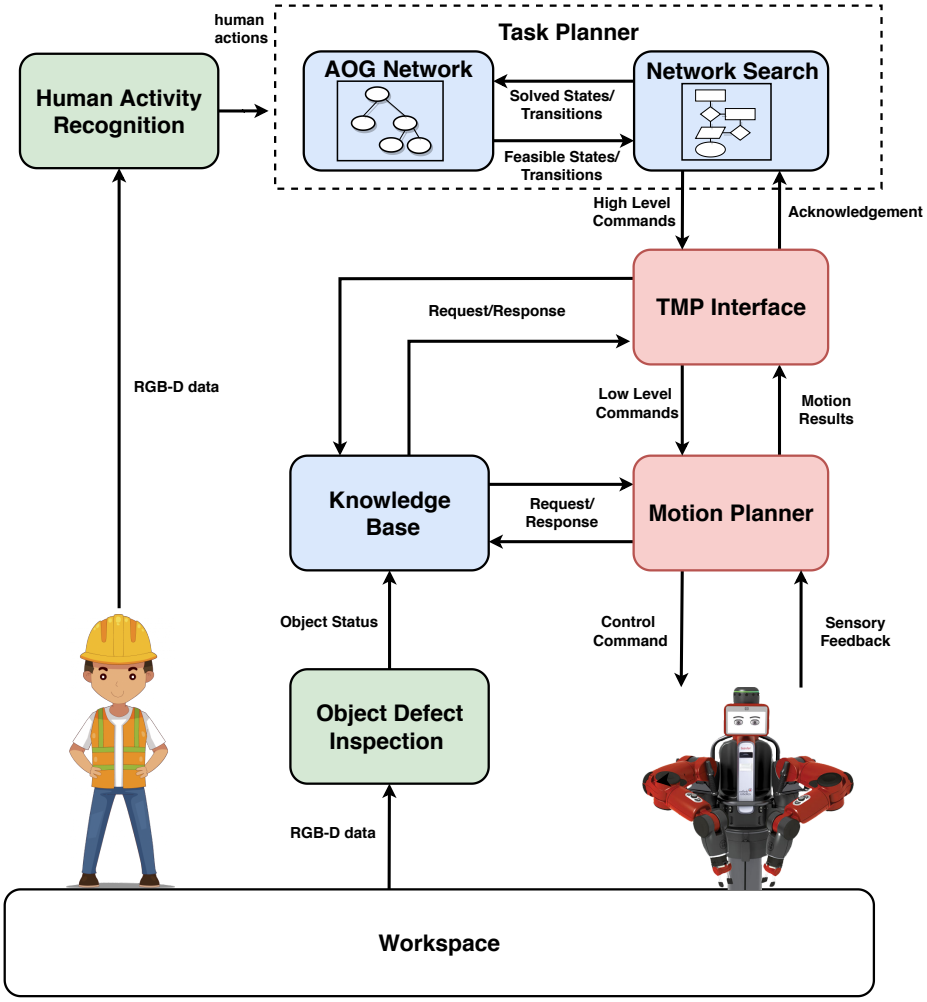


Figure 5.3 The system architecture integrating branched AND/OR graphs in a collaborative scenario.

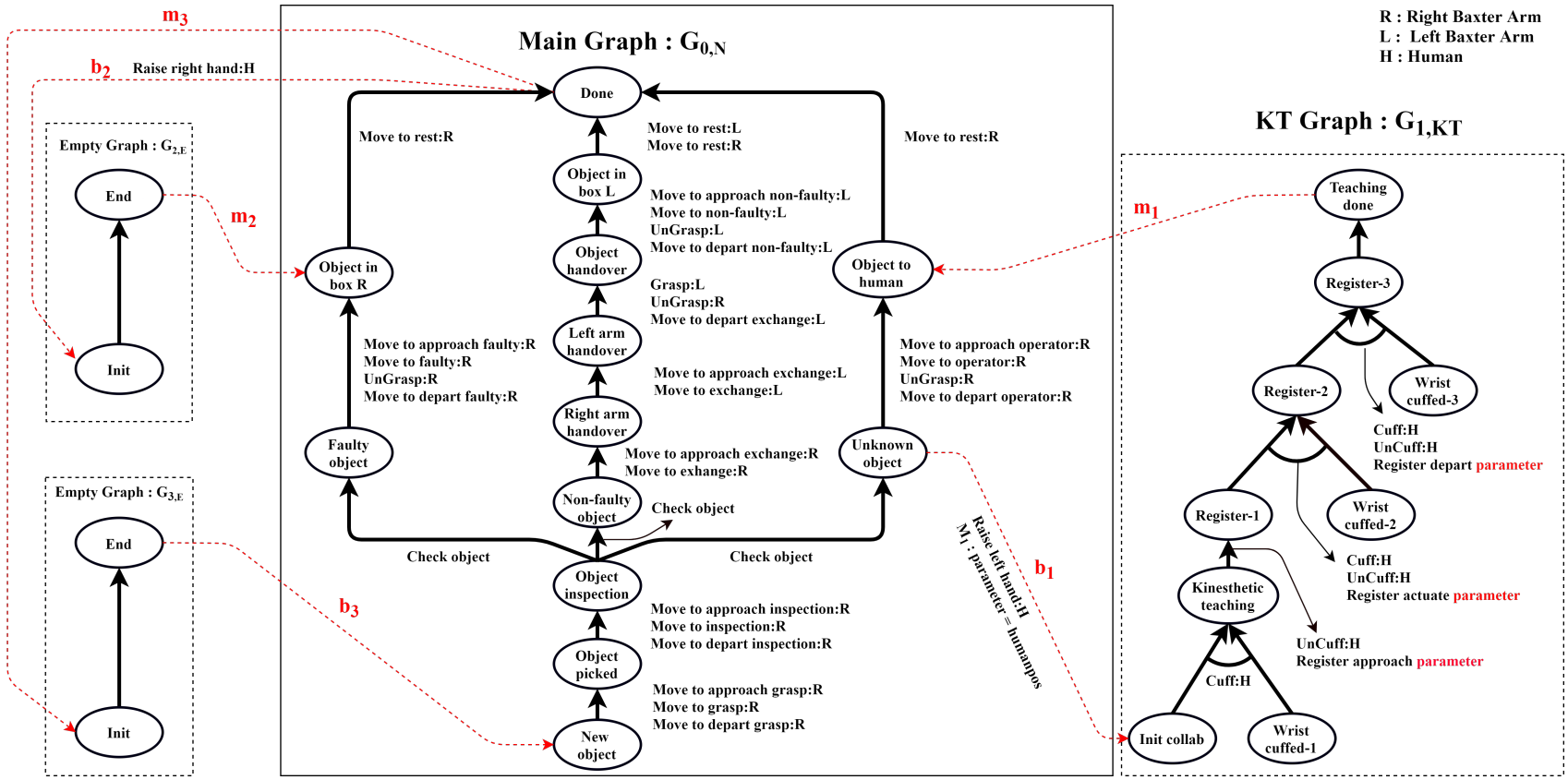


Figure 5.4 An example of an AND/OR graph network with three branchings. On the left, two Empty Graph branches ($G_{1,E}$ and $G_{2,E}$), and, on the right, one Kinesthetic Teaching Graph branch ($G_{1,KT}$).

5.3.1 Perception

In our application, the perception layer is composed of two modules, namely *Human Activity Recognition* (HAR) and *Object Defect Detection* (ODD). The HAR module collects data from an RGB-D camera to detect the operator's skeleton using the open-source software OpenPose [10]. The operator's skeleton is then processed to recognize two gestures, i.e., *raise left arm* (Figure 5.6b) and *raise right arm* (Figure 5.6o). Instead, the ODD module uses data from an RGB-D camera to classify objects into three classes: faulty, non-faulty and unknown. Since defect spotting is not the core of our work, this procedure simply uses AR tags glued on the objects.

5.3.2 Action

This layer comprehends two modules: the *Task Motion Planning Interface* (TMPI) and the *Motion Planner* (MP). The TMPI receives discrete, symbolic commands from the representation layer (e.g., PICK OBJECT A) and transforms them into grounded commands. Instead, the MP checks the grounded commands feasibility, sends the low-level commands to the robot and monitors the command execution acknowledging the TMPI once it is completed. For more details on how the action layer processes are handled, the reference is [21].

5.3.3 Representation

The Representation layer groups up the *Task Planner* (TP) and the *Knowledge Base* (KB) modules. The last one simply stores all relevant information about the robot's work space and the human as described in [54].

Instead, the TP groups up the AOG network (AOGN), storing the plan description; and the Network Search (NS) module, handling the decision making. The NS module collects the list of actions associated with each hyper-arc in the AOG and decides which actions should be executed, and who should perform it (human, left robot arm, right robot arm). The next action choice is based on the AND/OR graph traversal procedure introduced in [21]. Once an action is completed, the NS module updates the AOG internal state.
The TP also handles the branching mechanism. In our scenario, we have defined only two kinds of branched AOG: the Kinesthetic Teaching Graph (KTG) and the Empty Graph (EG, see Figure 5.4). The human operator can use KTGs to update relevant poses in the task (e.g., grasping, release, or inspection poses) while the robot is executing the plan. Instead, EGs

Algorithm 8: Branched AOGs algorithm

Data: G_N, G_{KT}, G_E ; /* Structure of normal, kinesthetic teaching and empty graphs */
Result: *Solved* G_N

```
1  AON ← ∅;                                  /* Create AND/OR graph network */
2  AON ← AON ∪ G_N;        /* AON.firstGraph is the main normal graph */
3  while true do
4      if Human gesture then
5          if !Scheduled then
6              if Right arm then
7                  G_E = createEmptyGraph();
8                  AON ← AON ∪ G_E;
9              else
10                 G_KT = createKTGraph();
11                 AON ← AON ∪ G_KT;
12             end
13         else
14         end
15         sendActionToTreeSearch(gesture)
16     end
17     if !Solved(AON.lastGraph) then
18         Solve(AON.lastGraph);      /* call algorithm 4, Task Planner */
19     else
20         if Solved(AON.firstGraph) then
21             break;
22         else
23             Solve(AON.firstGraph);    /* call algorithm 4, Task Planner
                */
24         end
25     end
26 end
```

allows the operator to traverse the graph backwards (e.g., to repeat a faulty action) or forward (e.g., to skip a non-necessary action).

The algorithm describing the branching mechanism is presented in the flowchart of Figure 5.5. As the process starts, the main graph G_0 (see Figure 5.4) is created and added to the graph network. This graph handles an inspection process in which the robot should: (i) grasp an object, (ii) classify it as faulty, non-faulty, or unknown, and (iii) release it in the correct position. Then the NS module, based on the AOGN's state, lists all the possible next actions and requests the action layer to check whether the robot can perform them.

If one or more actions from a hyper-arc are not feasible, the entire hyper-arc is disregarded, and the execution proceeds to the one that follows. When the NS module can not find any feasible action, a branching is activated, adding to the AOGN an EG $G_{i,E}$ (see Figure 5.4). The last node to be met in G_0 is set as the branching node n_{bi} and the leaf of $G_{i,E}$ is set as the parent of b_i. At the same time, the root of $G_{i,E}$ is set as the child of m_i. In this way, the merging node n_{bi} coincides with the branching node n_{mi} (i.e., branch and merge are performed on the same node of G_0).
This mechanism is necessary to compensate for errors in the action layer check. In fact, on the same actions, the check result can vary if repeated multiple times (i.e., the robot inverse kinematic solver uses random seeds). When all the actions associated with a hyper-arc get a positive check result, the corresponding state transition is selected for actuation. The graph traversal continues until the root node of G_0 is reached. After the root node is reached, when a new object is introduced in the scene, the system creates a new EG branch having as branch node the root of G_0 and as the merging node the only leaf of G_0. This branching resets the main graph, and since G_0 is designed to inspect only one object, it is necessary to process new objects.

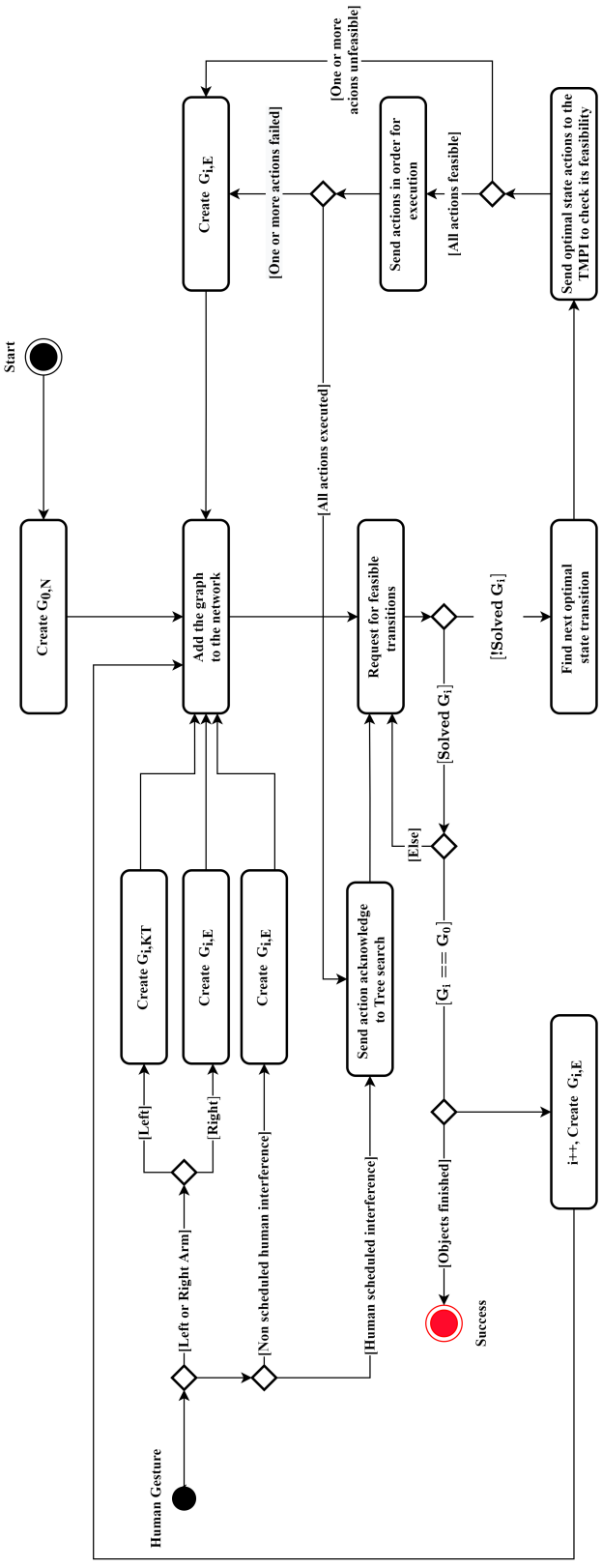


Figure 5.5 A flowchart describing the algorithm that the *Task Planner* uses to manage the branching mechanism.

At the same time, the framework allows the operator to branch G_0 with KTGs, to update plan poses, and with EGs, to repeat the actions associated with the last solved hyper-arc. As we previously described, the HAR module monitors the human operator's activities and recognizes two gestures *raise left arm* and *raise right arm*. As it is shown in the flowchart of Figure 5.5, when the human operator performs these two gestures, a new branch is generated respectively to a KTG or an EG. An abstract description of branched AOGN's is described in algorithm 8.

In our application, when a new KTG ($G_{i,KT}$) is created, the branching node n_{bi} is always the last node in G_0 that has been marked as met and the merging node n_{mi} is its parent in G_0. This operation mode is adopted because the actions defined in G_0, necessary for the transition from n_{bi} to n_{mi} to occur, are performed by the operator using kinesthetic teaching. The operator can use the KTGs to update relevant poses for the task (e.g., where the object should be grasped or placed). To guarantee a proper execution each relevant pose is composed of three sub-poses, namely *approach*, *actuate* and *depart*.
For this reason, our implementation of the KTG (see Figure 5.4) has three *Register* states, one for each of the three sub-poses. After the branching, when the operator grasps the robot wrist (*Cuff* action in Figure 5.4), the teaching process starts and, when the operator releases the robot wrist (*UnCuff* action), the new sub-pose is saved. To navigate the KTG and finish the teaching, the operator repeats these actions three times (see Figure 5.4).

At the same time, if the robot fails to perform an action, the operator can request its repetition branching to an EG. The branching associated with the *raise right arm* gesture creates an EG branching from the last met node to its child, allowing for repeating the last executed actions (see G_2 in Figure 5.4).

The operator can use the *Cuff* and *UnCuff* actions only while traversing the KTG. If one of these actions is detected in another context, it is considered an anomaly and the algorithm creates a new EG branch. A branch generated by this mechanism behaves like the one for the *raise right arm* gesture and allows the system to restore the plan execution ignoring operator interference's.

5.4 System demonstration

We demonstrate the validity of our approach using the dual-arm Baxter robot from Rethink Robotics, equipped with standard grippers and an RGB-D camera mounted on the robot's *head*. Baxter has a Zero-Gravity mode activated by grasping the arm's cuff (i.e., *Cuff*), allowing the operator to use kinesthetics to guide Baxter's arms. Baxter operates on a table

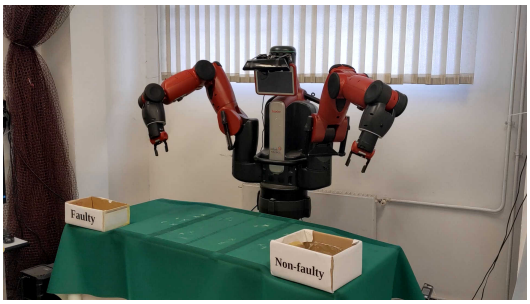
where objects to inspect are placed, and the human operator works on the opposite side of the table (see Figure 5.6). The experimental setup comprehends a second RGB-D camera positioned behind the operator. The robot camera inspects the object while the second one monitors the operator gestures.

We developed the architecture presented in the previous section using the Robot Operating System (ROS) framework [75], and the modules are implemented using either Python or C++. The architecture is distributed between two machines. The HAR module is executed on a workstation equipped with an Intel(R) Core i7-4790@3.6 GHz CPU, an NVidia GTX 970 GPU and 32 GB of RAM, while all the other modules run on a workstation equipped with an Intel(R) Core i7-8700@3.2 GHz CPU and 16 GB of RAM.

5.4.1 Setup Description

As anticipated, the considered scenario foresees a human operator and a robot assessing if an object presents defects. The robot waits in its initial configuration (Figure 5.6a) and the collaboration, described by G_0 (Figure 5.4), starts when the human places the first object on the table (*grasp pose*, Figure 5.6j). Then the robot grasps the object with its right arm and lifts it in front of its camera for inspection (*inspection pose*).
If the object is recognized as faulty, the robot reaches the box on its right-hand side, and releases the object (*faulty pose*, Figure 5.6l). If the object is non-faulty it has to be located on the box on the robot's left hand side (*non-faulty pose*, Figure 5.6n). However, that pose is out of the work space for the right arm, and the two arms meet in front of the robot (*exchange pose*, Figure 5.6m) to exchange the object and complete the task. Finally, if the object is classified as unknown, the robot should handle the object to the operator (*operator pose*, Figure 5.6k) to ask for her/his intervention.

All the poses necessary for the task execution are predefined at the beginning of the collaboration. However, during the execution, it could be necessary to update them (e.g., the object initial pose changes, the operator moves, or the boxes are re-allocated). In these cases, the operator can branch the main graph, performing the left arm gesture, with a KTG.
This mechanism allows the operator to update a specific pose using kinesthetic teaching. Since, as described previously, each relevant pose is composed of three sub-poses (approach, actuate and depart), the operator would have to teach all these sup-poses before the execution of the main graph is restored. The sub-poses taught by the operator are updated on the KB for later use. Instructions are displayed on the tablet mounted on the robot's head to guide the operator in traversing the KTG.


Faulty
Non-faulty

(a)

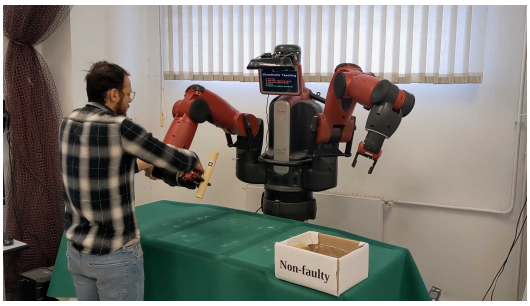
(b)


Non-faulty

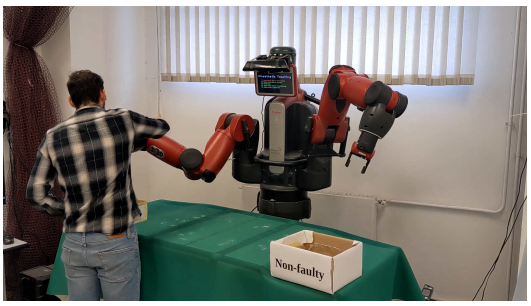
(c)


Non-faulty

(d)


Non-faulty

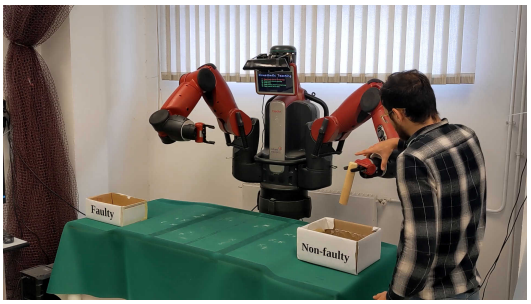
(e)


Non-faulty

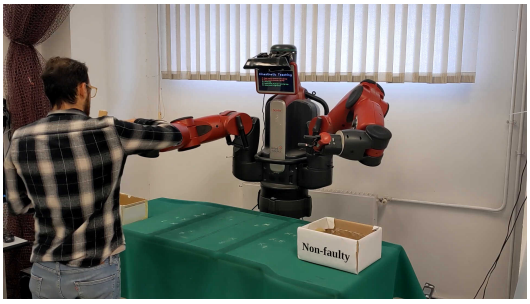
(f)


Non-faulty

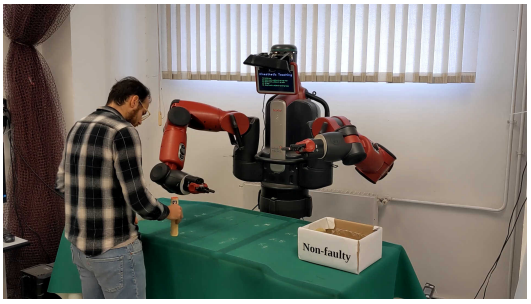
(g)


Faulty
Non-faulty

(h)


Non-faulty

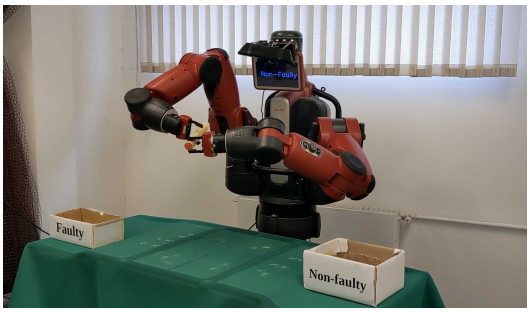
(i)


Non-faulty

(j)

(k)

(l)



(m)

(n)

(o)

(p)

Figure 5.6 Different stages of the Human-Robot Collaboration in which the operator alters the execution of the plan online.

Figure 5.6b shows the operator raising his left arm while the robot is moving to pick the object. This, activates a branching to update the *grasp pose*. A branched graph, $G_{1,KT}$, is created and the teaching process starts. The operator, by grasping a Baxter's cuff, moves forward in $G_{1,KT}$ to start kinesthetic teaching (Figure 5.6c). By releasing the cuff, the position of the end-effector is registered as *approach grasp pose* and the state *Register-1* in $G_{1,KT}$ is solved. The graph moves forward for all the following teaching steps (Figures 5.6d and 5.6e). Once the operator successfully solves $G_{1,KT}$, the *Task Planner*, moves back to the main graph G_0. Figure 5.6 presents also examples in which the operator teaches *faulty pose* (Figure 5.6f), *exchange pose* (Figure 5.6g), *non-faulty pose* (Figure 5.6h) and *operator pose* (Figure 5.6i).

It may happen that, because of errors in motion planning or motion execution, the robot fails the execution of one or more actions. In this case, the operator may raise his right arm (Figure 5.6o), requesting the robot to go back to the previous state to repeat the faulty actions. As explained in the previous section, this is possible by branching the main graph to an EG. Figure 5.6p shows one of these cases: the robot fails to release the faulty object in the box, and the operator requests to repeat the action by raising his right arm.

5.4.2 Results

A single operator repeated the inspection of one object for 10 times [2]. On average during each run the architecture created 6 different branches. Of these branches, an average of 3.1 have been activated by the operator, while the others are automatically generated by the system.

Table 5.1 Execution time for different modules of our architecture.

Module	Avg. time (s)	Std. dev (s)	Avg. Time (%)
AOG Network	0.02	0.001	0.08
Network Search	0.22	0.01	0.78
Motion Planning	1.62	0.41	5.63
Motion Execution	26.93	7.02	93.5
Total	**28.79**	**7.18**	**100**

The times related to the execution of each component are presented in Table 5.1 and include, average and standard deviation time over all the 10 runs. The AOGN time includes the time to create new branches that is negligible respect to normal AOGN operations. As

[2] www.youtube.com/watch?v=b9n0xZ6z9KE

shown in the Table, the processes associated with the representation layer (AOGN and NS), takes less than 1% of total average time. Also the standard deviation associated with the representation layer is negligible, pointing out that our framework is quite stable. The major portion of processing time is dedicated to motion planning and execution. It is worth noting that we did not include in the table the time that the operator spent to teach new poses. This has been done to avoid considering human proficiency with kinesthetic teaching.

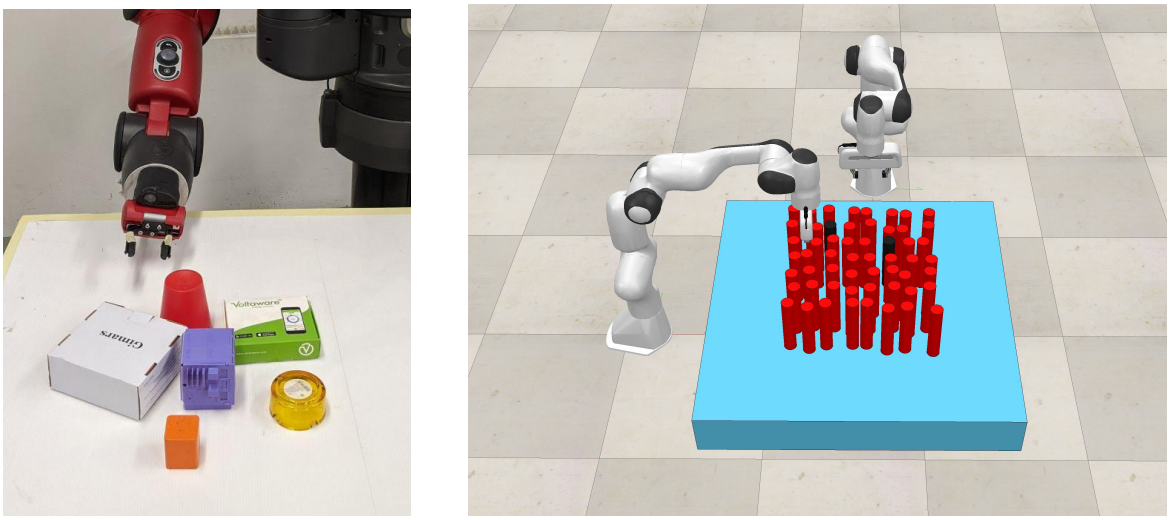
Chapter 6

Task and motion planning in robotic problems

6.1 Introduction

In common situations, humans trivially perform complex manipulation tasks, such as picking up a tool from a cluttered toolbox, or grabbing a book from a shelf, by re-arranging occluding objects. For humans, these tasks seem to be routine, and they neither require much *conscious* planning nor cognitive focus during action execution. Yet, for robots, this is definitely not the case. Such complex manipulation tasks as *picking from clutter* or *rearrangements* require advanced forms of reasoning to decide which objects to pick up or re-arrange, and in which sequence, so as to synthesize motions towards the target objects to account for the geometry-level feasibility of the task-level actions. This interaction between task-level, symbolic reasoning and geometry-level, motion planning is the subject of integrated *Task and Motion Planning* (TMP) [60].

The *de facto* standard syntax for specifying task-level actions is the Planning Domain Definition Language (PDDL) [67], and most approaches resort to it. However, task-level domains may also be specified using temporal logics [44], or formal languages [17]. Given the task-level domain, task planning finds a discrete sequence of actions from the current state to a desired goal state, which is expressed in symbolic form [38]. Likewise, motion planning approaches find collision-free configurations to reach a desired goal configuration, represented in geometrical terms [63]. Therefore, TMP approaches aim at establishing an appropriate mapping between task-level and motion-level domains. Examples of such a mapping include, given a task-level plan, the corresponding motion-level actions, or given a task-level state, the corresponding sequence of feasible geometric configurations.



(a) (b)

Figure 6.1 (a) The gripper needs to pick the purple cube. Only side grapes are allowed and the gripper is forced to act along a fixed plane to mimic a 2D scenario. (b) Cluttered table-top with two target objects (in black) and other objects in red. The multi-robot system consists of two Franka Emika manipulators.

Though many off-the-shelf PDDL-based planners are available, establishing the correspondence between task-level planners and motion planners is not easy a task, and requires the development of a full-fledged robot planning and control architecture. For the sake of the argument, let us consider a simple *cluttered table-top scenario*, as shown in Fig. 6.1a, where a robot must pick up a purple cube from clutter. As seen in the Figure, the target object cannot be immediately grasped, and other objects hindering the target grasp need to be properly re-arranged. Such a scenario may be modeled using two PDDL actions, namely pick and place. Obviously enough, in a TMP formulation, each task-level, symbolic action should also realize appropriate motions. Therefore, the typical PDDL domain may contain the following predicate definitions:

```
(:predicates (clear ?x) (gripper-empty) (holding ?x)).
```

The predicates (clear ?x), (gripper-empty), and (holding ?x) check if an object, generically identified with the variable ?x, is clear, i.e., nothing hinders its grasping, if the gripper does not hold any object, and which object the gripper is holding, respectively. In order to keep the scenario simple enough, we deliberately make the assumption that if the target grasp were hindered, the robot would be able to pick up another object, or otherwise different objects in sequence, to make the target pick action feasible. We note that PDDL-based planners initiate

the search through a process called *grounding*, which is aimed at replacing the variables in the predicates (for example the variable ?x above) with possible referents to real-world objects, so as to instantiate predicates and action schemas. If our scenario would comprise 6 possible objects to grasp and/or re-arrange, that would give rise to 13 possible propositions, i.e., 6 possibilities for each of the predicates (clear ?x) and (holding ?x), as well as one for (gripper-empty). Therefore, this would yield $n = 2^{13}$ possible states since we consider Boolean truth values. Finding the shortest path in the search space using a state-transition graph via a typical pick and place schemes would be characterized by a $O(n \log n)$ temporal complexity [8]. Heuristic search avoids visiting large regions of the transition graph via informed search to reach the goal state faster. However, it must be noted that *effective* goal reaching also depends on the configuration space, and that it should be verified that each pick and place action should be feasible at the motion level. However, what is even more relevant for this discussion is that, in a general sense, the number of objects that must be re-arranged to pick up the target object cannot be known beforehand, which would require observation-based re-planning within the PDDL framework, and in cascade an additional mapping between the PDDL-based action space and the robot observation space [6]. While PDDL planning is EXPSPACE-complete and can be restricted to less compact propositional encodings to achieve PSPACE-completeness [45], it is noteworthy that the time taken for each re-planning would further exacerbate the temporal complexity.

Although *single-robot* TMP has been an area of active research, current approaches do not naturally account for the benefits afforded by the presence of *multiple* robots. These may include the fact that tasks may be decomposed in a variety of ways, task allocation may involve different robots, or that re-arrangements may benefit from an explicit coordination among robots. A straightforward extension of single-robot TMP approaches to the multi-robot case would have to treat the multi-robot *system* as a collection of (possibly many) single robots, which would further worsen computational and temporal complexity as the number of robots would increase.

As a case in point, we consider a use case in which objects in a cluttered table-top setting must be re-arranged with the aim of reaching a target object, and we specialize the use case in two scenarios, that is with one robot only and with multiple robots, as in Fig. 6.1a and Fig. 6.1b, respectively. In the latter case, task allocation among available robots must be carried out, which is followed by a TMP method to carry out allocated tasks. In particular, in this paper we present a probabilistically complete approach for TMP in single and multiple robot scenarios. We tackle the two challenges described above, that is (i) the computational complexity of typical PDDL-based planners is quadratic, and (ii) the number of objects

to be re-arranged may not be known in advance, by defining the task-level domain using an AND/OR graph, and by encoding the task-level abstractions of the TMP problem in an efficient and compact way within the AND/OR graph. In view of challenge (ii) above, we introduce a variant of AND/OR graph networks able to iteratively deepen at run-time till reaching a state whereby the target object is grasped. With respect to challenge (i), in Section 7.1 we show that with our formalization the complexity of task-level planning is *almost linear* with respect to the number of graph iterations, that is, in our scenario, the number of objects that must be re-arranged. We then extend our TMP approach to leverage multi-robot capabilities: first, we allocate tasks to the available robots, and then we plan via AND/OR graphs a sequence of actions for the multiply decomposable tasks, which is optimal with respect to a multi-robot wide utility function.

6.2 Related works

In the recent past, TMP has received considerable interest among the Robotics research community [51, 85, 18, 31, 91, 34]. The primary challenge of planning in a hybrid, symbolic-geometrical space is to obtain an efficient mapping between the discrete task and the continuous motion levels. A combined search in the logical and geometric spaces using a state composed of both the symbolic and geometric variables is performed in [9]. A hierarchical approach for TMP is introduced in [50] wherein, planning is done at different levels of abstraction, thereby reducing longer plans to a set of feasible, shorter sub-plans. However, the planner plans backwards from the goal (that is, regression), and assumes that the actions are reversible while backtracking. A similar approach is also employed in [73, 26] to compute discrete actions with unbounded continuous variables.

Semantic attachments are used in [28, 27, 18, 91], associating algorithms to functions and predicate symbols via procedures external to the planning logic. Though semantic attachments allow for mapping between the task and motion spaces, all the relevant knowledge about the environment is assumed to be known before-hand. Besides, the robot configuration and the grasp poses need to be specified in advance, which renders the continuous motion space to be finite. The FFRob approach, described in [31], performs task planning via a search over a finite set of pre-sampled poses, grasps and configurations. This *a priori* discretization is relaxed in [49, 85, 92]. For example, the approach in [85] implicitly incorporates geometric variables while performing symbolic-geometric mapping using a planner-independent interface layer.

Most approaches (for example [29, 18]) first compute a task-level plan and refine it until a feasible motion plan is found or until timeout. To this end, the two approaches in [29, 18] adopt constraint-based task planning to leverage ongoing advances in solvers for Satisfiability Modulo Theories (SMT) [24]. The approach in [91], however, checks for the motion feasibly as each task-level action is expanded by the task planner. But this approach assumes a pre-discretized motion space and thereby a finite action set. Recent work of Caelan *et al.* [33] address this limitation by introducing *streams* within PDDL, which enable procedures for sampling values of continuous variables and thereby encoding an infinite set of actions.

However, the above mentioned methods do not consider the implication of having multiple robots in task allocation and collision avoidance, and would have to treat the multi-robot system as a combined single-robot system, which becomes intractable as the number of robots increases. Current TMP approaches for multi-arm robot systems focus on coordinated planning strategies, and consider simple pick-and-place or assembly tasks. As such, these methods do not scale to complex manipulation tasks [76, 95].
TMP for multi-arm robot systems in the context of welding is considered in [5], whereas [95] discusses an approach for multi-arm TMP manipulation. However, the considered manipulation task is a simple pick-and-place operation involving bringing an object from an initial position to a goal position, whereas two tables and a cylindrical object form the obstacles. A centralized inverse kinematics solver is employed in [69]. Motion planning for a multi-arm surgical robot is presented in [74], although predefined motion primitives are considered. Yet, a fine tuning of such primitives towards real experimental platforms remains a challenge.

TMP for multi-robot systems has not been addressed thoroughly, and therefore the literature is not sufficiently developed. Henkel *et al.* [46] consider multi-robot transportation problems using a Task Conflict-Based Search (TCBS) algorithm. Such an approach solves a combined task allocation and path planning problem, but assigns a single sub-task at a time and hence may not scale well to an increased number of robots.
Interaction Templates (IT) for robot interactions during transportation tasks are presented in [70]. The interactions enable handing over payloads from one robot to another, but the method does not take into account the availability of robots and assumes that there is always a robot available for such an handover. Thus, while considering many tasks at a time this framework does not fare well since a robot may not be immediately available for an handover.
A distributed multi-robot TMP method for mobile robot navigation is presented in [90].
However, they define task-level actions for a pair of robots and therefore optimal solutions are available for an even number of tasks, and only sub-optimal solutions are returned for an

Considered method	Task allocation	Task decompostion	Motion planning	Unknown number of sub-tasks
TCBS [46]	✓			
IT [70]	✓	✓		
[90]	✓		✓	
TMP-CBS [71]	✓	✓	✓	
Our	✓	✓	✓	✓

Table 6.1 Comparison of different multi-robot TMP methods.

odd number of tasks. Motes *et al.* [71] present TMP-CBS, a multi-robot TMP approach with sub-task dependencies. They employ a CBS method [81] in the context of transportation tasks. Constructing a conflict tree for CBS requires the knowledge of different constraints which depend on the sub-task conflicts, for example, two robots being present at a given location at the same time. However, in the table-top scenario considered in this paper, the number of sub-tasks is not known beforehand. The capabilities of the discussed multi-robot TMP methods are summarized in Table 6.1.

6.3 Task-motion formalism

Task planning or classical planning is the process of finding a discrete sequence of actions from the current state to a desired goal state [38].

Definition 1. *A task domain Ω can be represented as a state transition system and is a tuple $\Omega = \langle S, A, \gamma, s_0, S_g \rangle$ where:*

- S is a finite set of states;
- A is a finite set of actions;
- $\gamma : S \times A \rightarrow S$ such that $s' = \gamma(s,a)$;
- $s_0 \in S$ is the start state;
- $S_g \subseteq S$ is the set of goal states.

Definition 2. *The task plan for a task domain Ω is the sequence of actions $a_0, \ldots, a_m$ such that $s_{i+1} = \gamma(s_i, a_i)$, for $i = 0, \ldots, m$ and s_{m+1} satisfies S_g.*

Motion planning finds a sequence of collision free configurations from a given start configuration to a desired goal [63].

Definition 3. *A motion planning domain is a tuple* $M = \langle C, f, q_0, G \rangle$ *where:*

- C is the configuration space;
- $f = \{0, 1\}$, for collision ($f = 0$) else ($f = 1$);
- $q_0 \in C$ is the initial configuration;
- $G \in C$ is the set of goal configurations.

Definition 4. *A motion plan for M finds a collision free trajectory in C from* q_0 *to* $q_n \in G$ *such that* $f = 1$ *for* $q_0, ..., q_n$*. Alternatively, A motion plan for M is a function of the form* $\tau : [0, 1] \rightarrow C_{free}$ *such that* $\tau(0) = q_0$ *and* $\tau(1) \in G$*, where* $C_{free} \subset C$ *is the configurations where the robot does not collide with other objects or itself.*

TMP combines discrete task planning and continuous motion planning to facilitate efficient interaction between the two domains. Below we define the TMP problem formally.

Definition 5. *A task-motion planning with task domain* Ω *and motion planning domain M is a tuple* $\Psi = \langle C, \Omega, \phi, \xi, q_0 \rangle$ *where:*

- $\phi : S \rightarrow 2^C$, maps states to the configuration space;
- $\xi : A \rightarrow 2^C$, maps actions to motion plans.

Definition 6. *The TMP problem for the TMP domain* Ψ *is to find a sequence of discrete actions* $a_0, ..., a_n$ *such that* $s_{i+1} = \gamma(s_i, a_i)$*,* $s_{n+1} \in S_g$ *and a corresponding sequence of motion plans* $\tau_0, ..., \tau_n$ *such that for* $i = 0, ..., n$*, it holds that (1)* $\tau_i(0) \in \phi(s_i)$ *and* $\tau_i(1) \in \phi(s_{i+1})$*, (2)* $\tau_{i+1}(0) = \tau_i(1)$*, and (3)* $\tau_i \in \xi(a_i)$*.*

Chapter 7

Iteratively deepened AND/OR graph networks

7.1 AND/OR graphs networks

Here we recap AND/OR graphs and reformulate them, to prepare them forAND/OR graphs networks definitions.
An AND/OR graph is a graph which represents a problem-solving process [12]. Below we provide a brief overview of AND/OR graphs; a detailed exposition can be found in [53].

Definition 7. *An AND/OR graph G is a directed graph represented by the tuple* $G = \langle N, H \rangle$ *where:*

- N *is a set of nodes;*
- H *is a set of hyper-arcs.*

For a given AND/OR graph G, $H = \{h_1, \ldots, h_m\}$, where h_i is a many-to-one mapping from a set of child nodes to a parent node. Let us revisit the toy example considered in Section 6.1 (Fig. 6.1). An AND/OR graph for this scenario is visualized as a sub-graph in Fig. 7.1 consisting of the nodes graspable, gripper-empty, object picked, target placed and object placed. Note that we currently ignore the current configuration node (green color) and the graph extending from the object placed node in the figure, details of which will be described later. Each node $n_i \in N$ of the graph G represents a high-level state, for example, object picked. To achieve the state object picked, the gripper must be empty, that is gripper-empty and the object should graspable. Thus the nodes graspable, gripper-empty and object picked are akin to the PDDL predicates clear ?x, gripper-empty, and holding ?x, respectively.

The hyper-arc induces a mapping from the child nodes gripper-empty and graspable to the parent node object picked. In that sense, a hyper-arc induces a logical AND relationship between the child nodes/states, that is, all the child states should be satisfied simultaneously to achieve the parent state. Similarly, a single parent node can be the codomain for different hyper-arcs h_i. These hyper-arcs are in logical *OR* with the parent node. Nodes without any successors or children are called the *terminal* nodes. The terminal nodes are either a success node, that is, target placed or a failure node, that is, object placed.

Let us again consider the toy example in Fig. 6.1. If the number of objects to be re-arranged are known, then an AND/OR graph can be constructed using the the above mentioned states and corresponding transitions. Yet, the number of object re-arrangements is scenario dependent and not known ahead of time. The AND/OR graph representation thus seems incompatible. However, we make the following observation— the abstractions defined for clutter scenario, that is, the states (nodes of G) and actions (hyper-arcs of G) remain the same irrespective of the number of object re-arrangements. Thus we can envision a graph G that expands online by repeating itself until the target is retrieved. Nevertheless, each iteration of G corresponds to a new work-space configuration (object arrangement) and to encode this aspect we augment G with a virtual node that represents the current work-space configuration.

Definition 8. *For an AND/OR graph $G = \langle N, H \rangle$, an augmented AND/OR graph G^a is a directed graph represented by the tuple $G^a = \langle N^a, H^a \rangle$ where:*

- $N^a = \{N, n^v\}$ *with n_v being the virtual node;*
- $H^a = \{H, H^v\}$ *with* $H^v = \{h_i^v\}_{1 \leq i \leq |H^v|}$.

The virtual node n^v is called the root node of the augmented graph G^a. Each virtual hyper-arc h_i^v induces a mapping between the virtual node and a node of the graph G. In our cluttered table-top scenario, the most trivial case corresponds to successfully grasping the target without any object re-arrangements. However, most often due to clutter, a motion plan to the target do not exist and objects need to be re-arranged to obtain a feasible plan. In general, motion planner failure can arise if a path indeed does not exist or because the planning time allotted was insufficient. In this work, we assume that sufficient time is allotted to the planner so that a motion planning failure implies objects obstructing the path to the target.

Remark 1. *If the target grasp is unsuccessful, that is, the motion planner fails, then at least one object need to be re-arranged to search for a new target-graspable path.*

Thus, following the root node, a success node is reached if the target placed task is a achieved and hence the graph is *solved*. Else, the graph is terminated at the failure node (object placed) following the re-arrangement of a single object and the graph is therefore *unsolved*. In our approach a single augmented AND/OR graph G^a represents such a problem solving process. It readily follows from Remark 1 that if the graph G^a is terminated at the failure node a re-attempt is to be made to achieve a grasp of the target object. Such an attempt can again lead to a *solved* or *unsolved* graph. Therefore to achieve the required objective, it is necessary to iterate G^a till a success node is reached, that is, G^a is solved. This *roll-out* of augmented AND/OR graphs give rise to an AND/OR graph network.

Definition 9. *An AND/OR graph network* Γ *is a directed graph* $\Gamma = \langle \mathscr{G}, T \rangle$ *where:*

- $\mathscr{G} = \{G_1^a, \ldots, G_{n'}^a\}$ *is a set of augmented AND/OR graphs* G_i^a*;*
- $T = \{t_1, \ldots, t_{n'-1}\}$ *is a set of transitions such that* $G_{i+1}^a = t_i(G_i^a),\ 1 \leq i \leq n'-1$.

where n' is the total number of graphs in the network. Alternatively, n' is also the depth of the network. Note that t_i is defined only if G_i^a is unsolved, that is, if the graph terminates at the failure node. This transitions G_i to a new augmented AND/OR graph $G_{i+1}^a = t_i(G_i^a)$, updating the root node of G_{i+1}^a to the changed work-space configuration. Thus for the pick and place scenario, we obtain an AND/OR graph network as shown in Fig. 7.1.
The root node is the virtual node that represents the current work-space configuration (green node, that is, current configuration). The object to be picked and placed is decided by a cost function, for example, the proximity to the gripper and therefore for a single pick operation we need to traverse only the nodes till object picked. If the target object is being held, it is then put-down and the graph terminates, else a new object is selected to clear the path to the target. Once the object is moved, a new graph is grown with the current configuration being the updated current work-space configuration taking into account the object movement.

Complexity comparison: In Section 6.1 we have discussed the computational complexity for the toy example in Fig. 6.1 modeled using PDDL. We now analyze the complexity aspects when the same example is modeled using our AND/OR graph network (see Fig. 7.1). Assuming sufficient time is allotted for the motion planner, in the worst case all the 5 objects need to be re-arranged leading to 5×5 states[1] as opposed to 2^{13} using PDDL (see Section 6.1).

[1] 5 nodes for each object as seen in Fig. 7.1 and 5 iterations due to 5 objects.

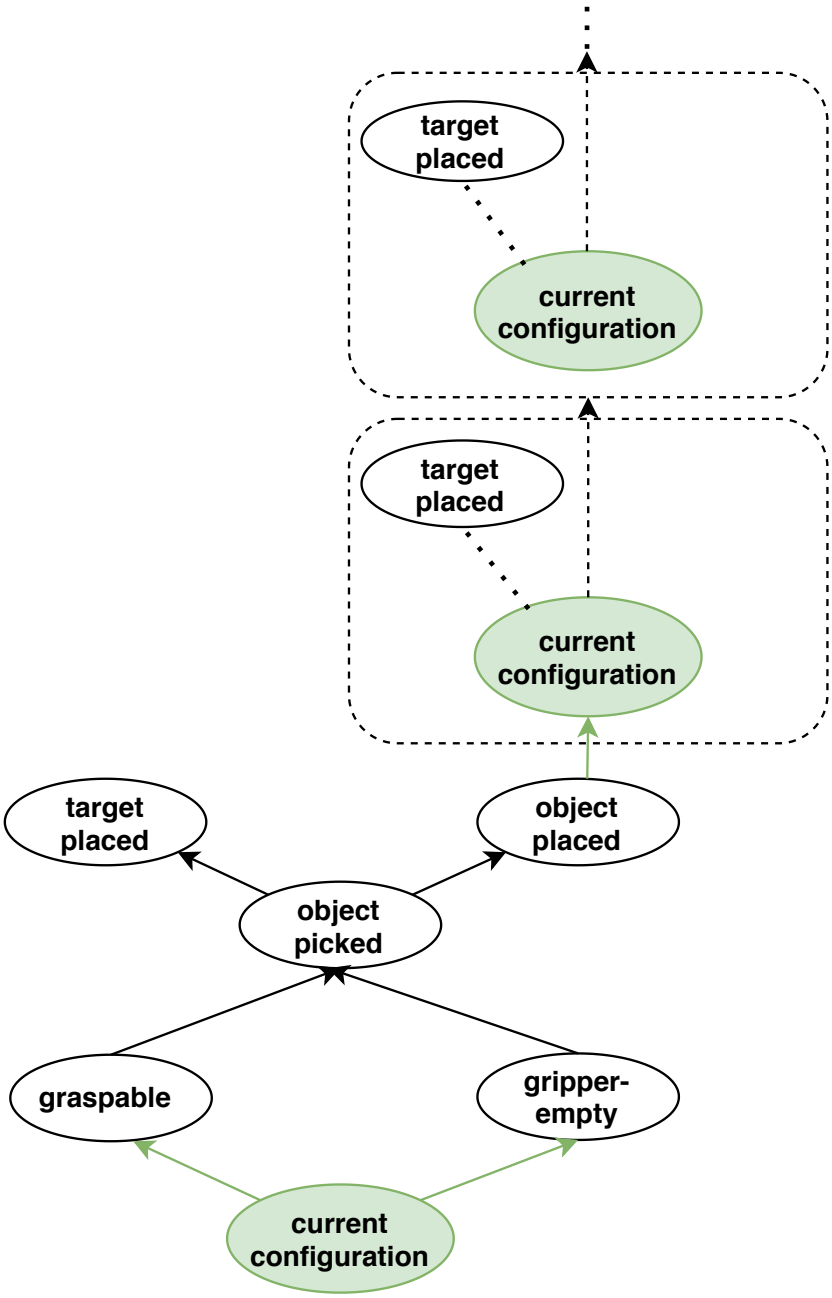


Figure 7.1 AND/OR graph network for the pick and place scenario shown in Fig. 6.1.

However it can happen that the task execution fails, for example due to motion, actuation or grasping errors. In such a case a new graph G^a_{i+1} is grown retaining the same work-space configuration as G^a_i. Therefore the number of iterations m may be greater than the number of objects (here $m > 5$) and hence the time complexity is thus $O(5m) \approx O(m)$. Note that for PDDL based planning, shortest path takes $O(n \log n)$ time where $n = 2^{13}$ is the number of states. In general for an AND/OR graph network with each graph consisting of n nodes, the time complexity is only $O(nm)$. Similarly, for an AND/OR graph with n nodes a storage of only $O(n)$ nodes is required.

Proposition 1. *An AND/OR graph network $\Gamma = (\mathscr{G}, T)$ is said to be solved at depth d when G^a_d is solved.*

Following Proposition 1, an AND/OR graph net with underlying augmented AND/OR graphs G^a_i is expanded till a G^a_i is solved. Note that as discussed in Section 6.1, we are interested in problems with solutions. For example in a cluttered table-top TMP scenario where a target object is to be grasped, we assume that a feasible grasp exist. This may result in a network of infinite depth. However, as a consequence of our assumption, that is, a solution exist, the network is solved at infinite depth. Moreover, to alleviate the issue of infinite depth, a predetermined depth limit may be employed. Thus, as the predetermined limit approaches infinity a solution is found. Practically, solutions with larger depth limits may be discarded as it may be too time consuming to arrive at. This enables us to define some sort of probabilistic completeness as will be seen later in Section 7.2.

7.2 TMP-IDAN

We present a novel TMP approach TMP-IDAN (Task-Motion Planning using Iterative Deepened AND/OR Graph Networks) that uses AND/OR graph networks to compactly encode the task-level abstractions to reduce the overall task planning complexity.

7.2.1 Single-robot TMP-IDAN

System's Architecture

An overview of TMP-IDAN is given in Fig. 7.2. In general the approach requires (1) *perception capabilities* to comprehend objects in the scene, (2) *task planning* to select the abstract actions and finally (3) *motion planning* for executing the action to manipulate objects, avoiding potential collisions. The perception capabilities are encapsulated in a single module, which is called the *Scene Perception*. This module provides the *Knowledge Base* module with the information about the current work-space configuration, that is, the location of objects and the configuration of robot. The planning layer is made up of two modules, namely the *TMP Interface*, and the *Motion Planner* modules. The *TMP Interface* module receives discrete or symbolic commands from the *Task Planner* and maps them to actual geometric values and drives the behavior of the *Motion Planner*. This module retrieves information regarding the work-space and robot from the *Knowledge Base*. It also provides an acknowledgment to the *Task Planner* upon the execution of a command by the robot. The *Motion Planner* module plans the outcome of robot behaviors before their actual execution.

The *Task Planner* module embeds the augmented *AND/OR Graph* and the AND/OR *Graph Net Search*. Along with the AND/OR graph, the *Task Planner* module is in charge of decision making and adaptation of the ongoing parallel (by parallel we mean different actions feasible from the current state) hyper-arcs. To do so, the *Task Planner* provides a set of achieved transitions between the states to the *Graph Net Search* module and receives the set of allowed cooperation states and transitions with the associated costs to follow. It then associates each state or state transition with an ordered set of actions in accordance with the work-space and robot configuration and finally incorporates the online simulation results to assign actions to the robot arms. Once an action is carried out, it receives an acknowledgment from the planning level and updates its internal structure. The *Knowledge Base* stores all relevant information to make the cooperation progress.

For motion planning, we use MoveIt [87], which supports RRT [59] from OMPL [88]. The motion planner is first employed to execute the obstacles selection algorithm. Once the tasks are allocated, the motion planner is called to (i) achieve the re-arrangement of each sub-task identified by the task planner and (ii) to grasp the target object.

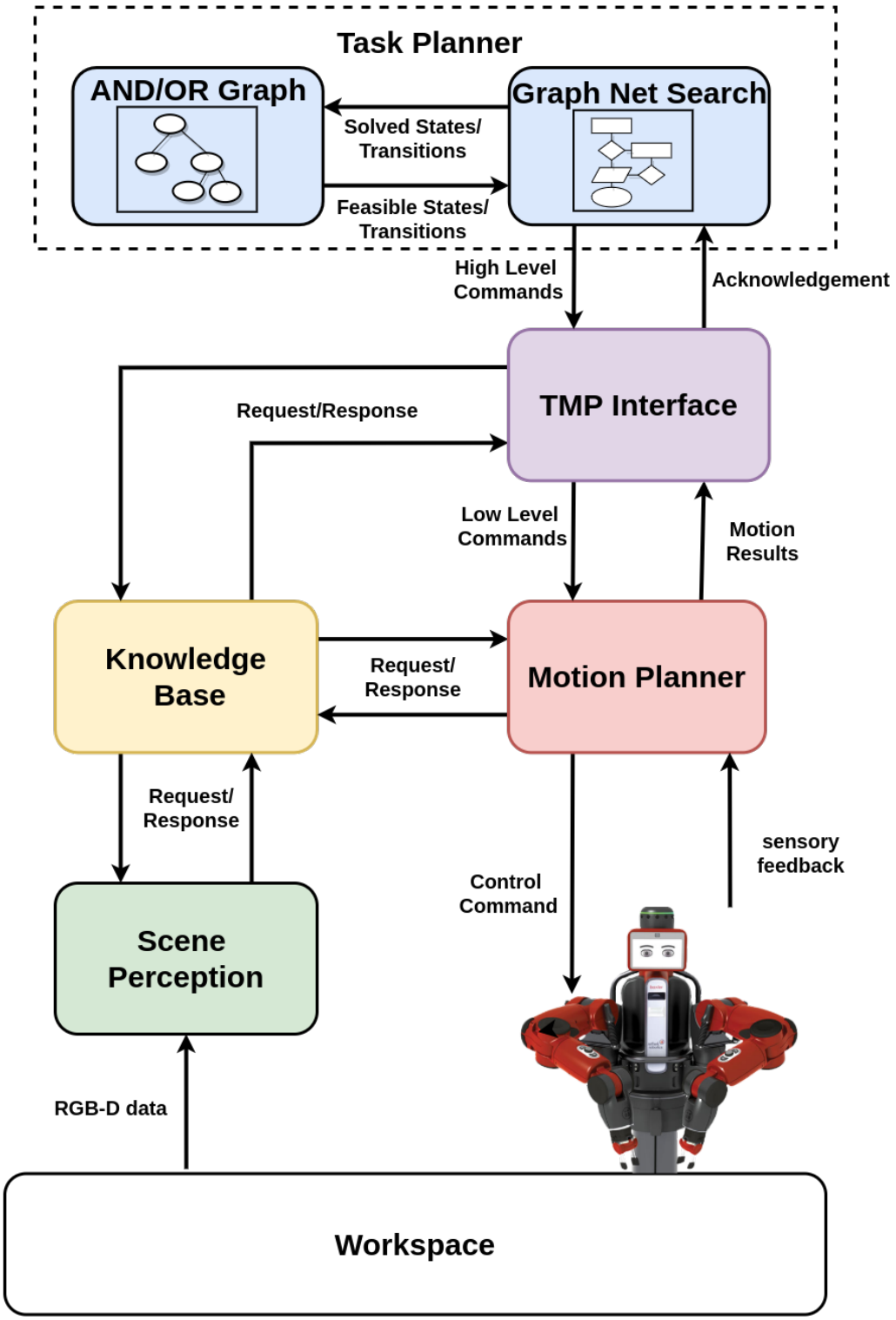


Figure 7.2 System's architecture of TMP-IDAN framework.

Task Planning with AND/OR Graph Networks

The AND/OR graph representation provides a framework for the planning and scheduling of task sequences [79]. The feasibility of the tasks is then checked using a suitable motion planner [53]. Moreover, an AND/OR graph inherently requires fewer nodes than the corresponding complete state transition graph, reducing the search complexity of the AND/OR space [79]. Yet, such a representation requires that the number of object re-arrangement to retrieve a target from clutter is known ahead of time. This representation thus seems incompatible as we do not know before-hand the number of objects to be re-arranged. To address this challenge, we introduce AND/OR graph networks as discussed in Section 7.1 wherein an augmented AND/OR graph grows online until the target grasp is achieved. For the cluttered table-top scenario, the AND/OR graph network with two graphs is seen in Fig. 7.3 (right). Given the initial work-space configuration, a virtual node representing the same (INIT#0) is added giving the augmented AND/OR graph G_0^a. The graph G_0^a encodes the fact that if a feasible grasping trajectory exists then the picked target task is to be performed and otherwise an object is to be identified to be either removed or pushed. Let us consider the case that a target grasping trajectory does not exit and that a grasped closest object of target followed by object placed in storage is achieved which is a failure node and thus exhausting G_0^a. This leads to a new graph G_1^a, which corresponds to a new augmented graph, with the same states and actions as G_0^a but a different work-space configuration encoded via the virtual node INIT#1. This process iteratively repeats itself until a graph $G_{n'}^a$ is solved which represents an AND/OR graph network Γ of depth n'. The augmented AND/OR graphs $G_0^a, \ldots, G_{n'}^a$ are thus iteratively deepened to obtain an AND/OR graph network whose depth n' is task depended and not known before-hand. We note here that in case a task in G_i^a fails, for example due to motion, actuation or grasping errors, G_{i+1}^a (with current work-space configuration) is grown and in this way our approach is robust to execution failures.

Algorithm 9 describes the overall planning procedure. It proceeds with the an AND/OR graph augmented with the initial work-space configuration (line 2) which is initiated by the AND/OR graph module via the call to *AddNewGraph*. The task planner module then checks for feasible states (call to *NextFeasibleStates*) in the augmented graph (line 3). From among the feasible states the optimal state is selected with the call to *NextOptimalState*. Our cost function is a combination of distance of the object to the robot base, distance to the left, right robot gripper and the size of the object. The tasks and agents (left and right robot arm) of the feasible state are communicated to the TMP interface with the call to *SendToTMPI*. Upon call to *RequestKB* and *RequestSP*, the geometric location of the objects, grippers

are communicated to the TMP interface though the knowledge base and scene perception modules. A motion plan is sought for by the TMP interface with the *RequestMP*. If the motion plan execution fails, for example a grasping failure, then a re-try is attempted (line 26). If a motion plan is not found then another object is selected for re-arrangement (line 9); see Remark 1. If motion plan is successful, a new graph is expanded and the process repeats until the target is retrieved.

Probabilistic Completeness

We now prove the probabilistic completeness of TMP-IDAN.

Lemma 1. *For a predetermined depth limit l, TMP-IDAN is probabilistically complete.*

Proof. For motion planning, we use RRT motion planner [59] which is probabilistically complete [52]. Thus, given sufficient time, the probability of finding a plan, if one exists, approaches one. For the AND/OR graph network $\Gamma = (\mathscr{G}, T)$ based task planner, by design each graph G_i^a terminates either at a success node or a failure node and hence each G_i^a complete. Using a predetermined depth limit l, Γ is expanded at most till depth l. Thus if G_j^a, $j \leq l$ is solved then Γ is solved. Else Γ is terminated at G_l^a and a non-existence of solution is reported. In reality, a solution does not exist at all or the chosen l is shallow. However, the first scenario (non-existence of a solution) is quashed since we are interested in TMP scenarios with solutions (consequence of our assumption). Thus, without any loss of generality, it can be argued that as l approaches infinity the probability of Γ being solved asymptotically approaches one. □

7.2.2 Multi-robot TMP-IDAN

We leverage the concepts detailed in the previous section and present a framework for multi-robot TMP. We begin by describing a heuristic approach providing a rough estimate of the objects to be re-arranged to pick a target object. The approach allows us to define a combined utility function for the multi-robot system to perform task allocation. Allocated tasks are then carried out using TMP-IDAN.

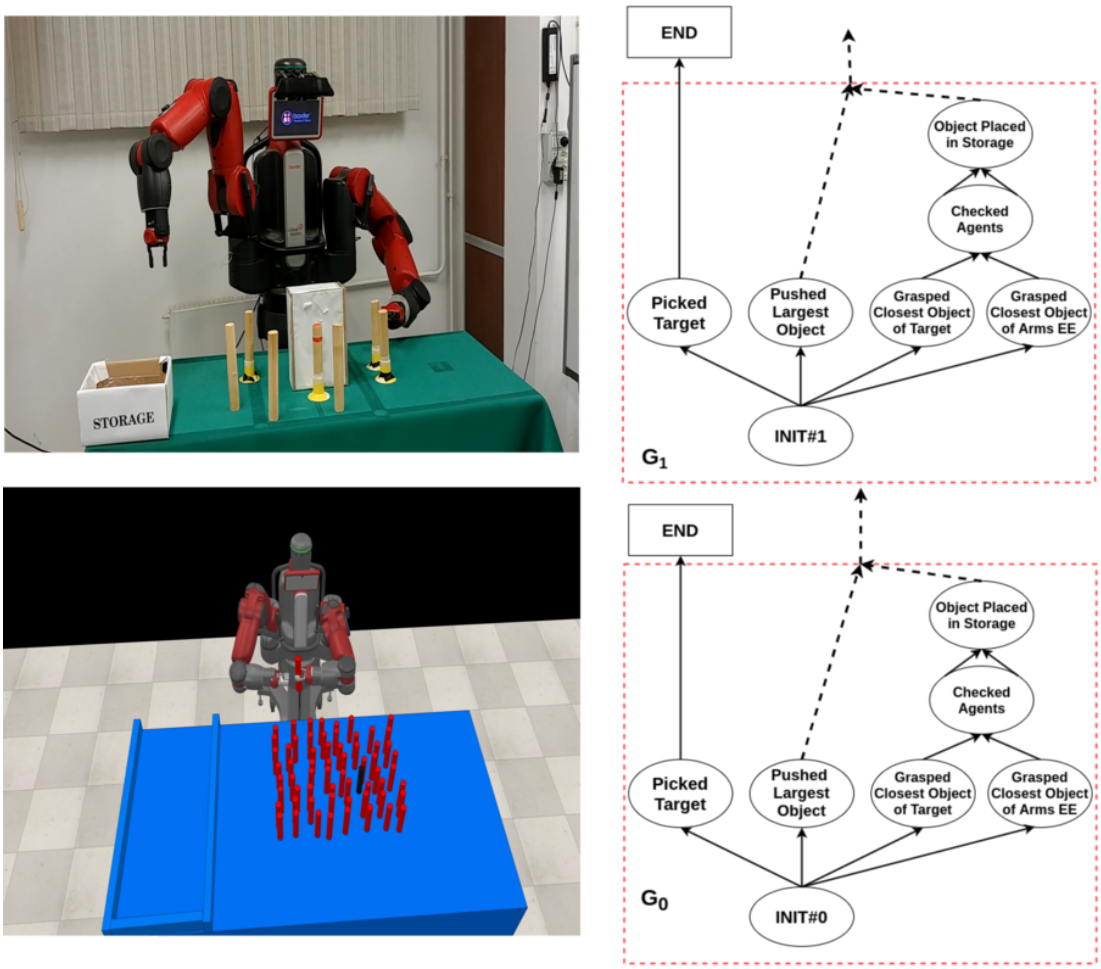
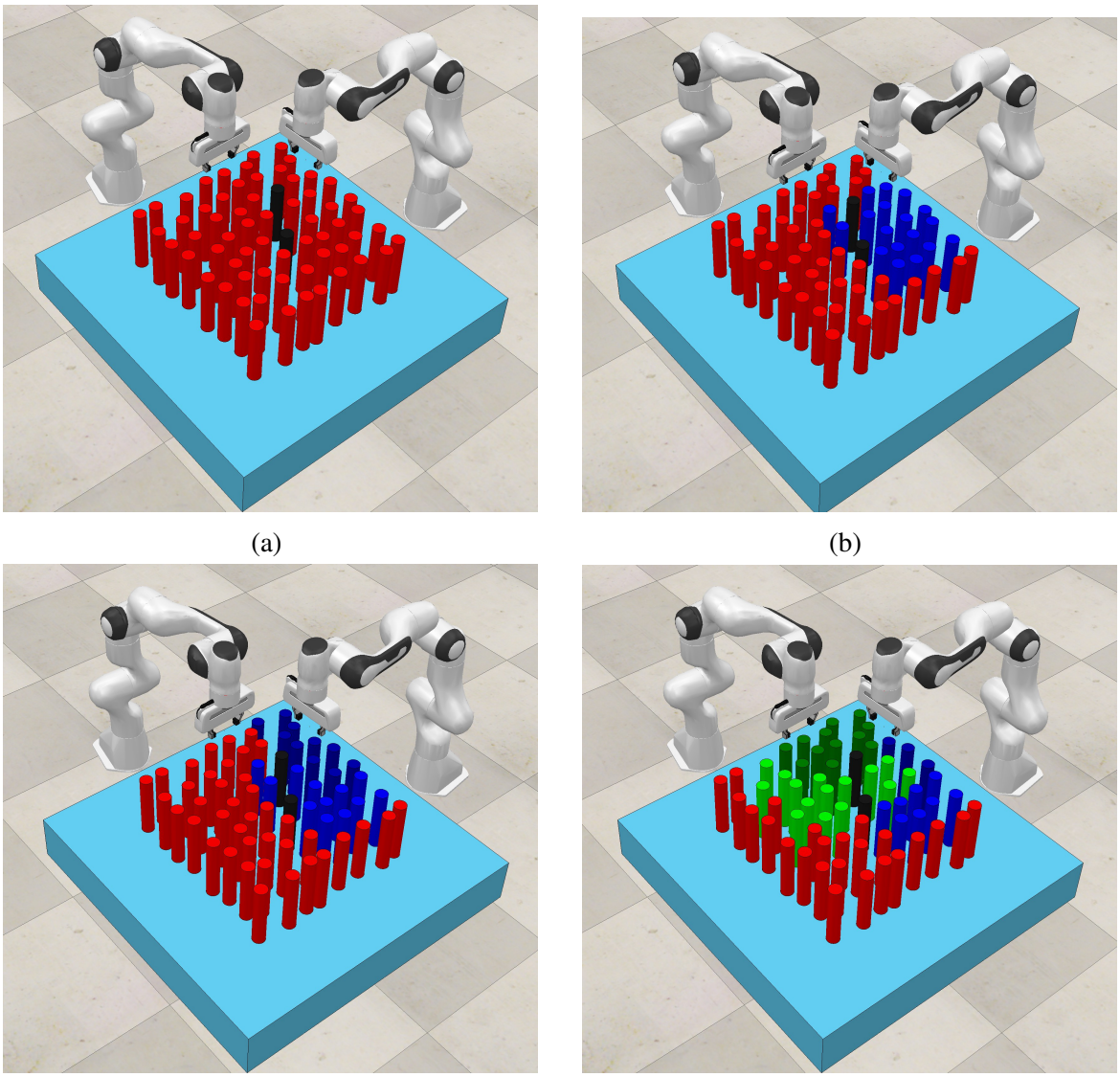


Figure 7.3 (top-left) TMP-IDAN in real world (bottom-left) and in simulation. (right) AND/OR graph network of depth 2 for the clutter scenario.



(a) (b)

(c) (d)

Figure 7.4 Illustration of the obstacle selection method: (a) a cluttered table-top scenario with two robots r1 (right), r2 (left) and the target objects in black; (b) a valid grasping angle range is computed by discretizing a fixed grasping angle range of $-\frac{\pi}{2}$ to $\frac{\pi}{2}$; the objects that fall within the grasping angle range of r1 (one target) are shown in blue; (c) blue objects within the grasping angle range of r1, considering both the targets; (d) Objects within the grasping range of both r1 (blue) and r2 (green), considering both the targets for each robot.

Obstacles Selection

An overview of the method can be seen in Figure 7.4. First, it finds different feasible plans to the target, each one corresponding to different grasping angles, by ignoring all the obstacles in the workspace. This is done by discretising the set of graspable angles, that is $\left[-\frac{\pi}{2}, \frac{\pi}{2}\right]$

(the axis is located at the center of the target).
It is noteworthy that we consider only side grasps to make the scenario more challenging. Among the available plans, the maximum and minimum grasping angles, namely α and β, are then obtained. The method then constructs two lines starting at the center of the target object and towards the robot with the corresponding angles α and β. The lines are terminated when there are no more obstacles along their paths. The end-points are then joined to form a triangle. This triangle is then enlarged on all the three sides by the radius of the bounding volume sphere of the end-effector. The objects within the constructed triangle are the objects to be re-arranged to facilitate the target grasp.

We note that what we describe here is an approximate method to identify the objects to be re-arranged. The actual set of objects depend on other such factors as the degrees of freedom of the robot, the size of the links and the end-effector, or the *degree* of clutter. As it will be discussed later, for all practical purposes we are interested only in an approximate measure so as to perform multi-robot task allocation.

Task Allocation

Let R be the number of available robots, and T the number of tasks to be allocated, that is, we have T target objects to be picked, such that $T \geq R$. Task allocation is performed offline and we assume that each task, that is, picking up a target object from clutter, is performed by a single robot, and that each robot is able to execute only one task at a time. We also recall here our assumption from the Introduction that the objects to be re-arranged to reach and pick a target are placed in a *safe* region. Our task allocation strategy falls under the Single-Task, Single-Robot, Time-extended Assignment (ST-SR-TA) taxonomy of Gerkey and Matarić [37], since the multi-robot system contains more tasks than robots. In order to allocate tasks, we define $U_{r_i t_j}$ as the utility function for a robot $r_i \in R$ executing a task $t_j \in T$. In this work, utility is inversely proportional to the number of object re-arrangements required to grasp the target object.
To determine such a measure, we first (randomly) select a target object t_1, and then for each robot r_i we run the obstacles selection algorithm described above. For each r_i, such a run returns the set of objects to be re-arranged to reach t_1.
Let us denote this set by $O_{r_i t_1}$ (and by $O_{r_i t_j}$ for the jth task). The robot r_k whose set $O_{r_k t_1}$ is of minimum cardinality (i.e., maximum utility) is then allocated task t_1. For the next target t_2 this step is repeated.

We note here that $O_{r_k t_2}$ is computed offline and returns the number of objects to be re-arranged by robot r_k to execute task t_2. However, it may be the case that some objects appear in both $O_{r_k t_1}$ and $O_{r_k t_2}$, that is, $O_{r_k t_1} \cap O_{r_k t_2} \neq \{\emptyset\}$. Since each robot executes one task at a time, r_k can execute t_2 only after having performed t_1.
Thus, during the execution phase, r_k may have already removed the common objects while executing t_1, and therefore these objects may be ignored to avoid *intra-robot* double counting while computing the utility $U_{r_k t_2}$ offline.

Once each robot is allocated a task, the set of remaining tasks T' is completed only after the execution of the assigned tasks. For the remaining T' tasks, the *inter-robot* or robot-robot double counting must be considered.
Reasoning in a similar manner for intra-robot double counting, the set $O_{r_i t_j}$ restricted to $T \setminus T' < j \leq T$ for the remaining T' tasks may have common objects with respect to $O_{r_i t_j}$ restricted to $1 < j \leq T'$ of the assigned tasks.
Thus, the total number of objects to be re-arranged for robot r_i to execute task t_j is

$$O^c_{r_i t_j} = |O_{r_i t_j}| - \sum_k |O_{r_i t_j t_k}| - \sum_k \sum_l |O_{r_i r_k t_j t_l}| \tag{7.1}$$

where $|\cdot|$ denotes the cardinality of a set,

$$O_{r_i t_j t_k} = \begin{cases} O_{r_i t_j} \cap O_{r_i t_k} & \text{if } r_i \text{ allotted } t_k \text{ previously,} \\ 0 & \text{otherwise.} \end{cases} \tag{7.2}$$

and

$$O_{r_i r_k t_j t_l} = \begin{cases} O_{r_i t_j} \cap O_{r_k t_l} & \text{if } r_k \text{ allotted } t_l \text{ previously,} \\ 0 & \text{otherwise,} \end{cases} \tag{7.3}$$

with the terms $|O_{r_i t_j t_k}|$ and $|O_{r_i r_k t_j t_l}|$ modeling the intra-robot and inter-robot double counting, respectively. We therefore have the following utility function

$$U_{r_i t_j} = \frac{1}{1 + O^c_{r_i t_j}}. \tag{7.4}$$

The maximum utility of $U_{r_i t_j} = 1$ is therefore achieved when no object re-arrangement is required to execute task t_j, that is, $O^c_{r_i t_j} = 0$. Using the taxonomy in [58], we thus have In-schedule Dependencies (ID) – the effective utility of an agent for a task depends on what other tasks that agent is performing as well as Cross-schedule Dependencies (XD) – the

effective utility of an agent for a task depends not only on its own task but also on the tasks of other agents.

We now define the combined utility of the multi-robot system, which consists of maximising

$$\sum_{i \in R} \sum_{j \in T} U_{r_i t_j} x_{r_i t_j} \tag{7.5}$$

$$\begin{aligned} &\text{such that} \sum_{i \in R} x_{r_i t_j} = 1 \\ &\text{where} x_{r_i t_j} \in \{0, 1\}. \end{aligned} \tag{7.6}$$

From (7.4) and (7.5) we see that the robot with the minimum number of object re-arrangements for a given task is thus assigned the maximum utility. In case of a tie, we select the robot which has not been allocated any task. If all the robots with the same utility have been allocated tasks already, or if none has been allotted, then a robot is selected randomly.

Task Decomposition

Each manipulation task is decomposed into a set of sub-tasks which correspond to pick-and-place tasks, that is, re-arrangement of the objects that hinder the target grasp.
As seen above, the number of sub-tasks for a given task are not known beforehand. Moreover, *multiple decomposability* [98] is possible since the clutter can be re-arranged in different ways.
We seek a decomposition minimizing the number of sub-tasks for the multi-robot system. This can be achieved during task allocation since utility is computed based on the obstacles selection method. In this work, we consider *complex task decomposition* [98] – a multiply decomposable task for which there exists at least one decomposition that is a set of multi-robot allocatable sub-tasks. Though in this work we ignore the multi-robot allocatability property, this can be incorporated trivially.
For example, let us consider the case where two robots have the same utility to perform a task t_j. In this case, the sub-tasks can be equally divided to achieve multi-robot allocatability or one robot may be selected randomly (or depending on previous allocation) to perform the entire task.

The Multi-robot Task-Motion Planning Loop

The overall system's architecture of our multi-robot TMP-IDAN method is similar to the single-robot architecture shown in Fig. 7.2 with the only difference being the *Network Search* module. Since we consider here two robots and their respective configurations need to be updated, we have two *Network Search* modules corresponding to each robot. In general, this would mean n such modules for an n robot system.
Once the tasks have been allocated, *Task Planner* selects the abstract actions whose geometric execution feasibility is checked by *Motion Planner*. The *Task Planner* layer consists of the *AND/OR Graphs* module – the initial augmented AND/OR graphs for the robots, and the *Network Search* module – the search procedure iterating the initial augmented graphs. As discussed previously, the initial augmented AND/OR graph consists of the task-level actions for each robot augmented with the current workspace configuration. *AND/OR Graphs* provides a set of achievable transitions between the states to *Network Search*, and receives the set of allowed states and transitions as the graph is expanded.
Task Planner then associates each state or state transition with an ordered set of actions in accordance with the workspace and robot configurations. The *Knowledge Base* module stores the information regarding the current workspace configuration, that is, the objects and their locations in the workspace as well as the robot configuration. This module augments the graphs with the current workspace configuration to facilitate *Network Search*. *TMP interface* acts as a bridge between the task planning and the motion planning layers. It receives action commands from *Task Planner*, converts them to their geometric values (for example, a grasping command requires various geometric values such as the target pose or the robot base pose), and passes them on to *Motion Planner* to check motion feasibility. To this end, the module retrieves information regarding both the workspace and robots from *Knowledge Base*.
If an action is found to be feasible, it is then sent for execution. Upon execution, *Task Planner* receives an acknowledgment regarding action completion and the *Knowledge Base* is updated accordingly.

In this section we validate TMP-IDAN in the context of both single and multiple robots. All the experiments are conducted on a workstation equipped with an Intel(R) core i7-8700@3.2 GHz $\times$ 12 CPU's and 16 GB of RAM. The architecture is developed using C++ and Python under ROS Kinetic.

7.3 Experimental Results

7.3.1 Single-robot TMP-IDAN

The experiments are performed on Rethink Robotics dual-arm Baxter robot which is equipped with standard grippers and an RGB-D camera mounted on its head and pointing downward which is used to acquire images for object detection. In order to validate the effectiveness of TMP-IDAN, we consider a cluttered table-top scenario wherein a target object is to be retrieved from among clutter (see Fig. 7.3(top-left and bottom-left)).

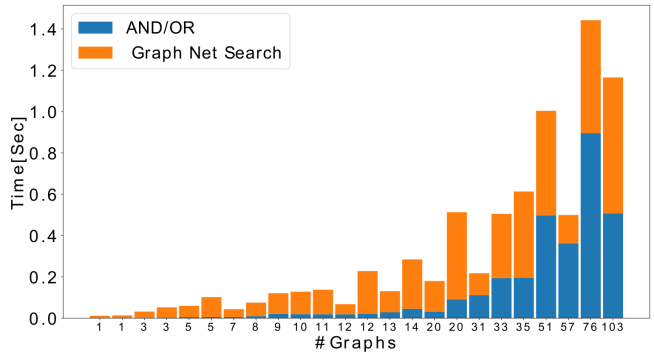
The experiment scenario includes two physical agents, that is, the right and left arms of the Baxter and a table-top with cylinders and cuboids. Removed objects are placed in a storage close to the right arm and in case the object is picked by the left arm, it is handed to right arm and finally placed in the storage. This sequence goes on until the target grasp is feasible.

Description of the experiments

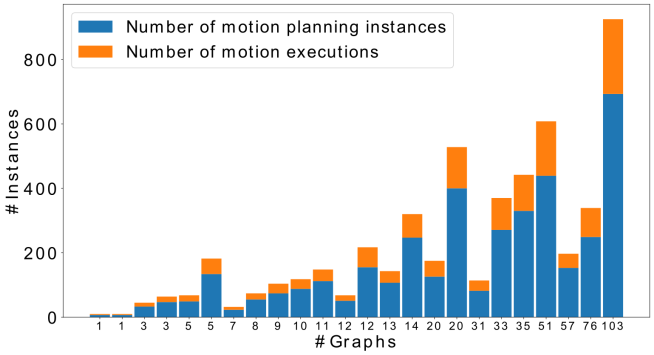
To verify the adaptability of TMP-IDAN in real-world scenarios and it's robustness with respect to grasping failures, we first employ TMP-IDAN in real world with Baxter robot. Subsequently we perform twenty four different experiments using the state-of-the-art robotics simulator CoppeliaSim [77] to corroborate that our approach is highly fast and has a linear computational time profile with respect to number of expanded graphs. Moreover, simulation environment enables us to induce scenarios where number of objects is not known in advance and also vary the target object location randomly. In simulation, we start with four objects in the scene and increase the complexity by adding up to 64 objects. In all the runs, the position of target object is chosen randomly. Fig. 7.3 (right) shows the AND/OR graph network employed. For the initial graph G_0 there exists several parallel hyper-arcs of which picking the target object is given the minimum cost. If the target is not reachable by any of the agents, then closest object to target is set to be removed and once a feasible hyper-arc is found, then agents move forward in the graph with the set of assigned actions.

Validation

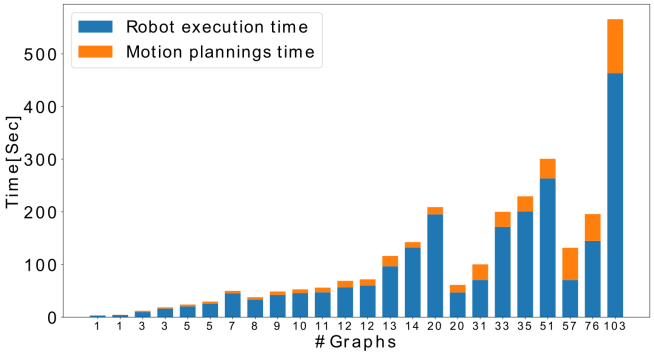
Table 7.1 shows the average planning and execution time for each module discussed in Section 7.2. We increase the number of objects on the table and for each object number, we perform the experiment 3 times by randomly sampling the target location. Table 7.2 reports the average network depth d, average total task planning time, average total motion



(a)



(b)



(c)

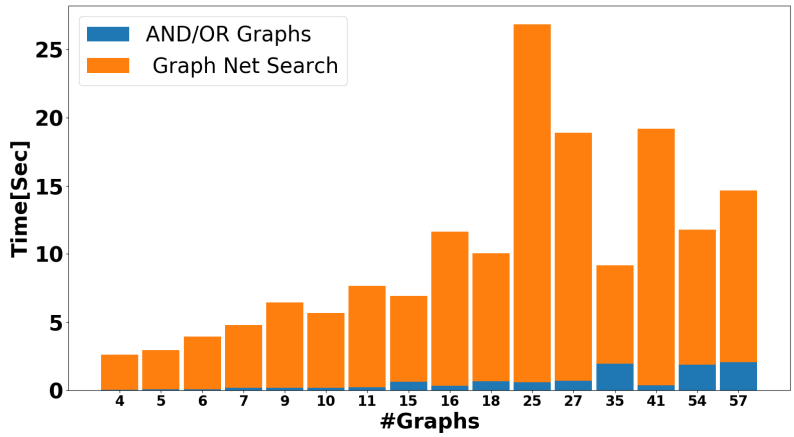
Figure 7.5 Various histograms with increasing network depth.

planning time, average number of motion planning attempts and the average number of objects to be re-arranged. Task planning times with increasing network depth corroborates our discussion on time complexity (see Section 7.1) and is almost linear with respect to d. For motion planning, we use Moveit package [87] as RRT planner within OMPL [88] and

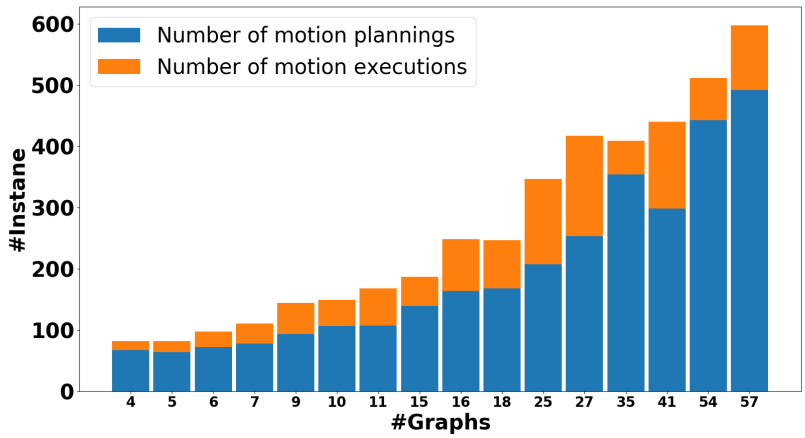
we place a time bound of one second. However, motion planning failures due to actuation errors, grasping failures or occlusion lead to re-plan, explaining the large number of motion planning attempts.
In addition to the computational complexity analysis in Table 7.2, Fig. 7.5 shows different histogram plots with increasing network depth d. Fig. 7.5(a) shows the total task planning time with d. As seen above, the planning times are almost linear with increasing d. However, slight deviations are readily observed. For example, the time for $d = 76$ is greater than the time for $d = 103$. This is due to the fact that in many cases, due to motion planning failure a new graph is expanded before reaching the terminal node. Thus for $d = 76$, more number of nodes are traversed when compared to $d = 103$ and rightly justifies the histogram. Fig. 7.5(b) reports the number of motion planning attempts and the total executions with increasing d. Note that this depends on the degree of clutter and thereby the number of object re-arrangements required. This is more clearly observed in the last two columns of Table 7.2 where increase in motion planning attempts can be seen with increased object re-arrangements. Finally, in Fig. 7.5(c), total execution times with d can be seen.

7.3.2 Multi-robot TMP-IDAN

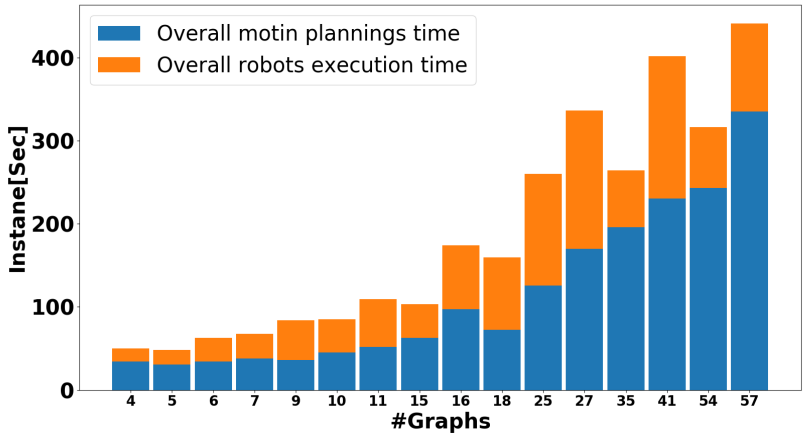
We validate and demonstrate the performance of multi-robot TMP-IDAN by performing experiments in the state-of-the-art robotics simulator CoppeliaSim [77], employing two Franka Emika manipulators. As seen in Fig. 6.1b, we consider a cluttered table-top scenario where manipulation tasks correspond to picking up different target objects. For each task, the sub-tasks consist of removing objects hindering each target grasp. In this work, such objects are picked and placed outside the working area, in a *safe* space. We begin with 6 objects in the work space with two of them being target objects. To test the scalability to an increasing number of objects, we perform experiments with up to 64 objects. For a given number of objects the experiment is conducted 3 times, and in each experiment the target objects are chosen randomly. Table 7.3 reports the average combined planning and execution times for robots r_1 and r_2, focusing on the architecture's modules. AND/OR graph and network search times are per number of grown graphs. It must be noted that motion planning failures due to actuation errors or grasping failures lead to re-plans, and therefore to a larger number of motion planning attempts. Table 7.4 and Table 7.5 report the average network depth d, the average total task planning time, the average total motion planning time, the average number of motion planning attempts, and the average number of objects



(a)



(b)



(c)

Figure 7.6 Various histograms with and increasing AND/OR graph network depth.

to be re-arranged for robots r_1 and r_2, respectively. As the AND/OR graph network depth d increases, task planning times are *almost* linear with respect to d. This is so because for an AND/OR graph network with each graph consisting of n nodes, the time complexity is only $O(nd)$. In contrast, PDDL-based planners are characterized by a search complexity of $O(n \log n)$, where $n \approx 2^{13}$ for the table-top scenario.

Fig. 7.6 shows different histograms for an increasing network depth d. In particular, Fig. 7.6(a) shows the total task planning time with d. One can readily observes that the linearity in planning time is not strictly followed. For example, the time for $d = 25$ is greater than the time for $d = 41$. In many cases, due to motion planning failures a new graph is expanded before reaching the terminal node, which implies that more nodes are traversed for $d = 25$ compared to $d = 41$, therefore explaining the variations. Fig. 7.6(b) plots the number of motion planning attempts and the total executions with an increasing d. An increase in d in most cases correspond to a higher degree of clutter. Therefore, as depth d or the graphs increase the motion planning attempts increases as well. However, motion planning failures can also increase the depth d since a new graph need to be expanded. This explains the slight deviation in the trend. Fig. 7.6(c) reports the total motion planning and execution times with an increasing d, and the plot readily follows from the discussions above.

Algorithm 9: TMP-IDAN

```
require : Augmneted AND/OR Graph (AO), Task Planner (TP), TMP Interface
          (TMPI), Motion Planner (MP), Knowledge Base (KB), Scene Perception
          (SP)
1  while Target not retrieved do
2      AO:AddNewGraph();
3      TP: Request AO NextFeasibleStates()
4          if Request = empty then
5              Go to Line 2;
6          else
7              continue;
8          end
9          Find NextOptimalState();
10         for tasks, agents in OptimalState do
11             SendToTMPI(tasks,agents);
12             for task in tasks do
13                 for agent in agents do
14                     TMPI: RequestKB();
15                     KB: RequestSP();
16                     TMPI: RequestMP();
17                 end
18             end
19         end
20     if RequestMP() then
21         TP: FindOptimalMotionPlan()
22             TMPI: T ← Optimal Motion Plan;
23             if T executed then
24                 Go to Line 3;
25             else
26                 Go to Line 2;
27             end
28     else
29         Go to Line 9;
30     end
31 end
```

Module	Average [s]	Std. Dev [s]
AND/OR graph	0.0031	0.0028
Graph network	0.0104	0.0048
Motion planning attempts	9.3980	5.6640
Motion executions	3.3540	1.9320
Motion planning time	0.8010	0.2170
Motion execution time	4.7490	2.0240

Table 7.1 Computation times for different modules of single-robot TMP-IDAN and their corresponding standard deviations.

Objects	d	TP [s]	MP [s]	MP attempts	Objects re-arranged
4	1.67	0.018	1.044	15.66	1.33
8	7.33	0.068	4.560	50.66	4
15	14.33	0.170	20.893	177	7.5
20	57	0.422	61.168	400	18
30	19.66	0.188	28.591	159.66	9
42	72	0.462	44.148	384.5	18.5
49	26.5	0.342	24.265	201	10
64	76	0.657	102.575	693	29

Table 7.2 d- average network depth, TP- average total task planning time, MP- average total motion planning time, average motion planning attempts and the average number of objects to be re-arranged as the degree of clutter is increased.

Activity	Average [s]	Std. Dev. [s]
AND/OR Graphs	0.03213	0.02174
Network Search	0.8992	0.1862
Motion planner (attempts)	18.320	6.450
Motion execution (attempts)	6.3126	0.8176
Motion planner (time)	0.8010	0.2170
Motion execution (time)	4.7490	2.0240

Table 7.3 Average computation times for different modules and their corresponding standard deviations.

Objects	d	TP [s]	MP [s]	MP attempts	Objects re-arranged
6	2.33	1.5505	16.339	36.66	1.66
8	2.66	1.422	13.721	32.66	1.66
9	2.33	1.3532	16.909	30.66	1.0
12	4	2.4025	20.3363	42.66	2.66
16	4.66	2.755	20.1718	47.66	3.33
20	3.66	2.125	17.618	42.33	2.66
30	5.75	3.464	25.854	56.75	4.25
49	17	8.642	81.361	147.0	6.33
64	22.6	9.053	132.179	203.0	12.2

Table 7.4 Legenda: d - average AND/OR graph network depth, TP - average total task planning time, MP - average total motion planning time. The average motion planning attempts and the average number of objects to be re-arranged as the degree of clutter is increased for robot r_1.

Objects	d	TP [s]	MP [s]	MP attempts	Objects re-arranged
6	2.33	1.1201	13.207	31.33	1.0
8	2.0	1.1478	18.154	28.66	1.0
9	3.66	2.415	18.520	40.0	2.33
12	3.33	2.0532	17.825	37.66	2.33
16	4.33	2.6326	22.830	45.66	3.33
20	6.33	4.022	25.182	61.33	5.33
30	9.25	4.748	45.226	83.75	5.75
49	12.33	6.886	49.593	106.66	10.66
64	12.6	5.271	66.994	109.875	7.0

Table 7.5 Legenda: d - average AND/OR graph network depth, TP - average total task planning time, MP - average total motion planning time. The average motion planning attempts and the average number of objects to be re-arranged as the degree of clutter is increased for robot r_2.

Chapter 8

Solving TAMP benchmarks

In chapter 7, we validated scalability and performance of *Iteratively deepened* AND/OR graphs in TMP-IDAN framework and proved the efficiency of this framework in two robotic TAMP scenarios. However, those scenarios do not represent a complete TAMP problem or do not address various challenges that may raise in different TAMP scenarios. In this chapter we describe these challenges and address them, implementing the same TMP-IDAN for all these challenges.
We selected five well-known TAMP benchmarks in the literature. Each of theses benchmarks bring a new hurdle within their complex structure. Nevertheless, AND/OR graphs, as proved throughout this dissertation are flexible enough to map any process into their underlying structure. In particular, Iteratively deepened AND/OR graphs, can easily model any process with unknown horizon.
TAMP benchmarks from [61] are given below.

- Tower of Hanoi [9, 42]
- Blocks-World 3D [19]
- Sorting Objects [85, 44, 32, 31, 33]
- Non-Monotonic [32, 31, 33]
- Kitchen [50, 85, 32, 31, 35]

Each of TAMP benchmarks for robotics problems, include one or more features that are either related to the specifications of motion planning, task planning and/or the interface of interaction among various levels of planning. In specific following features are included:

- Blocking objects

- Large task spaces
- Infeasible task operators
- Trade-off motion planning/task planning
- Non-monotonicity
- Non-geometric actions

Table 8.1, demonstrates better the criteria included in each of these benchmarks [61].

Criteria	TH	BW	SO	NM	KT
Infeasible task actions	✓	✓	✓	✓	✓
Large task spaces	✓	✓	✓		
Motion/Task Trade-off		✓			
Non-monotonicity				✓	✓
Non-geometric actions					✓

Table 8.1 Criteria for each TAMP benchmark, TH- Tower of Hanoi, BW- Block-World 3D, SO- Sorting Objects,NM- Non-Monotonic, KT- Kitchen

Here we shortly explain aforementioned criteria. **Infeasible task actions** refer to actions that are not explicitly feasible mainly due to unfeasible motion plans, i.g., obstacles occluding target objects or position of target object is exceeds the kinematic limits of the robot.
Large task spaces imply that underlying task planners requires a lot of effort for searching and this is mainly because there are numerous type of objects that extends the horizon of reasoning.
Motion/Task Trade-off means that less task planning search is due to adaptable motion planning and more task planning search relaxes constraints on motion planning space, thus potentially less precise for grasping and object pick/placement.
Non-Monotonicity is dividing a single goal to many sub-goals and repeat sub-goals unless ultimate goal is reached. For instance to pick a target object from a cluttered table, some objects need to be arranged several times so that target object is picked.
Eventually, **Non-geometric actions** are actions that are made of primary actions, such as cooking, washing and cutting a cabbage.

These benchmarks are independent of type of planners for both task and motion planning attributes and mainly focuses on computational complexity of the planners and relaxes many robotic real-life implementation limitations through following assumptions [61].

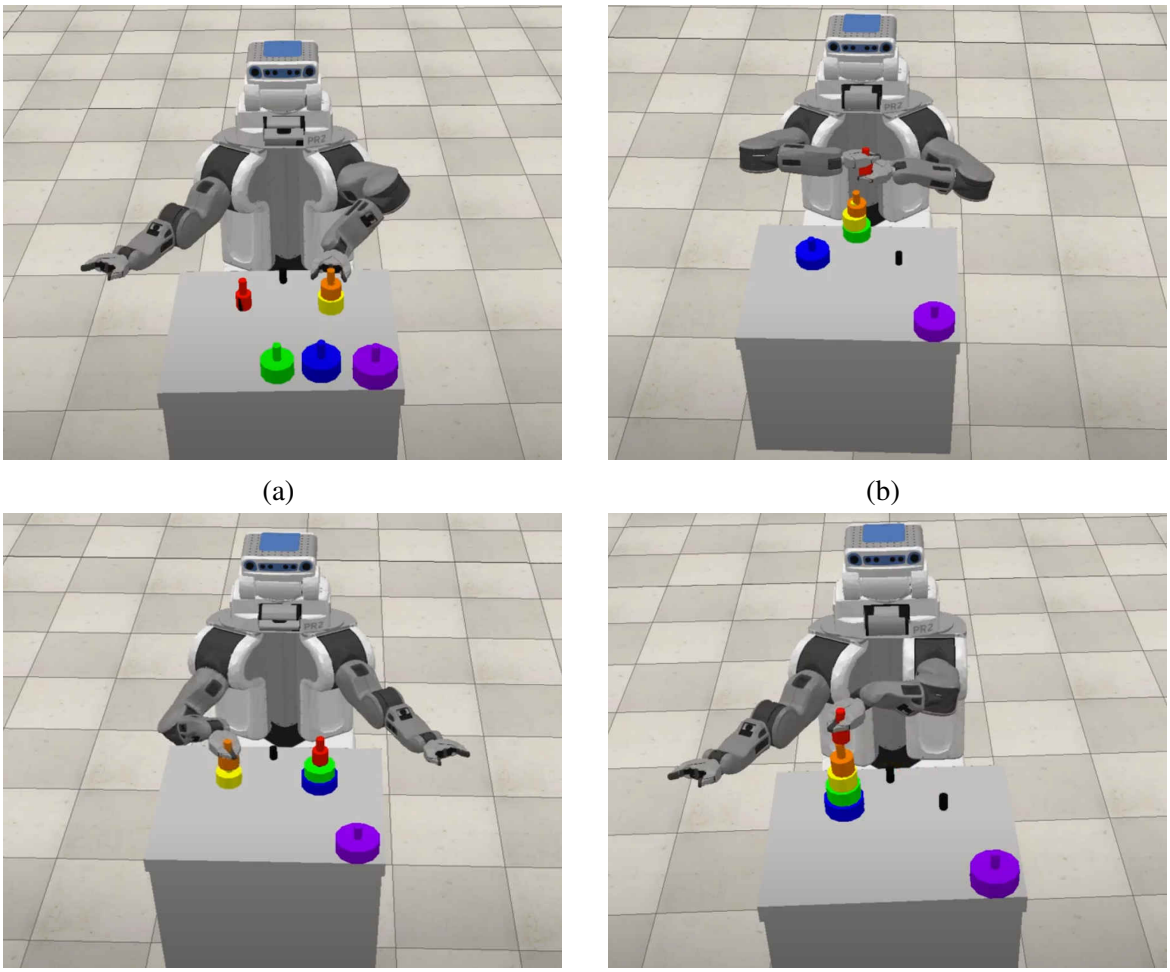
1. **Geometric:** Motion planning over positions only is sufficient. Objects that are grasped or placed are kinematically coupled with the parent object and do not move or slide and are static.

2. **Fully observable:** Initial states are known both geometrically and semantically. This applies for both robots and objects in the environment. This assumption can be extended to the perception level, where geometric status of all involved elements are known during the process.

3. **Deterministic:** All the robot motions are object pick/place operations follow the motion planner. Therefore errors entailed due to the stochastic behaviours of motion planners are neglected and their influence on the overall process are considered.

8.1 Towers of Hanoi

We revisit the classic problem of towers of Hanoi, with three rods that are placed in triangular formation. This formation creates hindrances for disk pick up and placements as *infeasible task action* criteria. Triangular formation of rods entail challenge for *large task space* criteria whereby, disks can not directly placed to their destination rod.
The objective of this problem is to move a pile of disks from one peg to a destination peg using intermediate peg. The optimal solution for n number of disks is $2^n - 1$ disk displacements. Disks can be only placed on bigger disks and at each step, only one disk can be moved. In our experiment we implement different number of disks varying from three to six disks. Figure A.1, depicts the AND/OR graph structure that we used to implement Hanoi problem. Off course this is a possible graph to solve this problem, one can propose their own way of solving this problem through AND/OR graphs. This graph compacts fifty nodes and fifty-three hyper-arcs. Active agents are right and left arms and their corresponding grippers while Base is fixed. Figure 8.2, illustrates depth of AND/OR network for various number of disks in blue bars. whereas red curve is the optimal solution for number of disk movements, $2^n - 1$. It is noteworthy to mention that each of AND/OR graphs in the network is implemented to move only one disk from one peg to another in case of success. Hence as the Figure 8.2 suggests, the average number of AND/OR graphs are almost two times more than optimal solution. We speculate that this is mainly because of two major reasons: 1) To the vast majority of the disk movement cases, due to obstacles and kinematic range limit of arms, robot is not able to move a disk from it's place to the destination and therefore has to use intermediate peg, which imposes adding an extra AND/OR graph to complete



(a) (b)

(c) (d)

Figure 8.1 Illustration of towers of Hanoi benchmark in simulation environment: (a) left arm picking disk from left peg to place it to intermediate peg, (b) left arm is hand-overing the disk to right arm due to it's kinematics limits in reaching to left peg (c) right arm is placing disk to the left peg (d) all five disks are placed to their destination peg successfully

the process. 2) In some cases the motion planner due to it's stochastic internal algorithm is not able to find a feasible trajectory, thus it might be another source to add extra AND/OR graphs.

Tables 8.2 and 8.3 reveal more information of the TAMP modules such as Tree search, motion planning attempts and times.

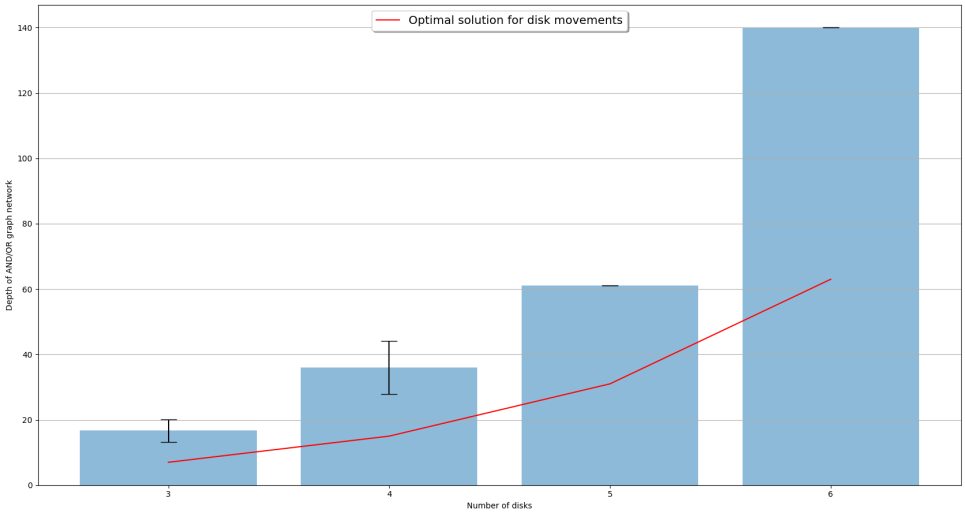


Figure 8.2 Average and standard deviation of number of AND/OR graphs for Hanoi problem for various number of disks

Objects	d	TP [s]	Right MP [s]	Right MP attempts	Left MP [s]	Left MP attempts
3	16.67	0.78	105.67	76.06	111.37	88.10
4	31	3.27	245.64	200.22	186.65	129.3
5	61	15.2	458.24	442.7	409.75	315.90
6	140	59.73	1017.30	854.15	678.68	475.5

Table 8.2 Legenda: d - average AND/OR graph network depth, TP - average total task planning time, Right/Left MP - average total motion planning time for arms. Right/Left MP attempts - The average motion planning attempts for arms r_1.

Objects	d	TP [s]	Right MP [s]	Right MP attempts	Left MP [s]	Left MP attempts
3	3.51	0.007	18.1	0.74	12.25	4.43
4	8.18	0.37	47.92	20.09	15.8	0.29
5	0	0.61	1.86	20.31	14.49	16.31

Table 8.3 Legenda: d - standard deviation AND/OR graph network depth, TP - standard deviation total task planning time, Right/Left MP - standard deviation total motion planning time for arms. Right/Left MP attempts - The standard deviation motion planning attempts for arms.

8.2 Blocks-World 3D

The objective of this benchmark is to stack all ten cube blocks in alphabetical order on the red tray, see figure 8.3. This benchmark exercises *task/motion planning trade-off*, *infeasible*

task actions and *large task spaces*. As the rules, cube hand-over between robot arms is not allowed, so one should either place few number of blocks on the table move them to another tray, or, place more number of blocks on the table and move the most significant cube(s) to another tray. The area on table for cube placement has to be limited to allow both arms to reach the cubes easily. Only arms are allowed to move and the base is fixed. On the top of the table there is shelf obstacle that limits forming pile of stacks on the table with more than two cubes. Moreover only top grasps are allowed for both cube pick and placement.

In our experiment we preferred to have a dens top-table rather than sparse table. We place as much as possible number of cubes on the table and move the most significant to other tray. Figure A.2 shows the structure of AND/OR graph we adopted to implement this benchmark. This AND/OR graph has 31 nodes and 35 hyper-arcs. Statistics of all modules are listed in Table 8.4. As table indicates, 74 graphs on average are required to sort 10 cubes on the blue tray. Consider that each graph grants only one cube displacement, either from tray to table or vice versa. Another import note that can be driven from the table is that right arm has been involved almost twice the left arm, and this makes sense, since the destination tray is next to the right arm. High value for standard deviation in network depth can be related to stochastic nature of the motion planner.

Table 8.4 Statistics of various modules implemented for Block-World 3D

Module	Average	Standard deviation
Depth of AND/OR network	74	12.72
AND/OR graph	10.30 [s]	0.44 [s]
Graph search	1.16 [s]	0.004 [s]
Right arm plan	471.3 [s]	0.18 [s]
Left arm plan	257.37 [s]	10.23 [s]
Right arm plan attempt	327.91 [s]	11.23 [s]
Left arm plan attempt	176.26 [s]	8.13 [s]

8.3 Sorting objects

The objective of this problem is to move all N blue blocks to the left table and all N green blocks to the right table. There are 2N red blocks as obstacles that occlude the table for reaching to blue and green blocks. This problem involves both the *infeasible task actions* and *large task space* criteria. The presence of red blocks makes task actions infeasible and

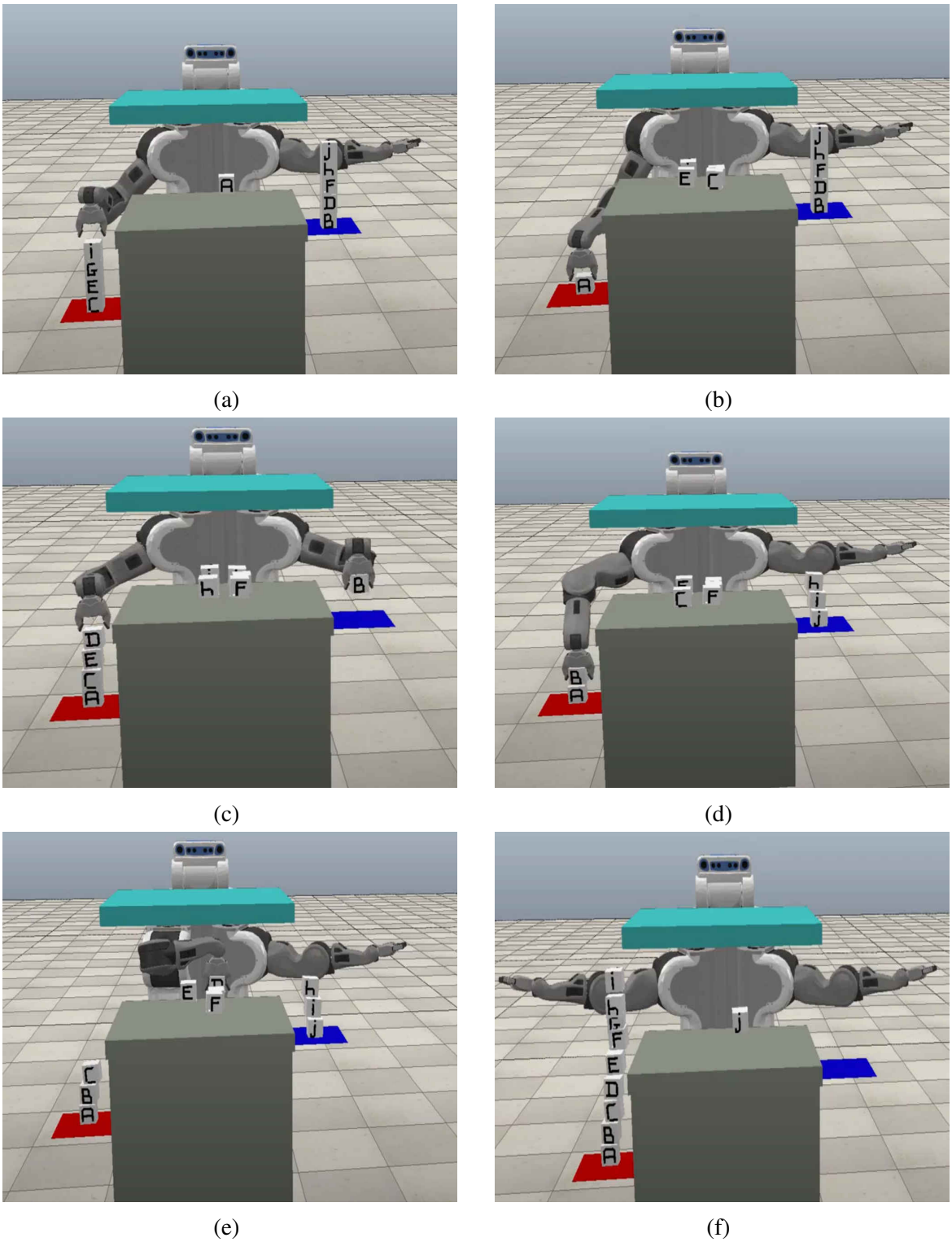


Figure 8.3 Illustration of blocks-world benchmark in simulation environment: (a) and (b) right arm evacuates all the cubes on the red-tray and places them on table, (c) and (d) while left arm evacuates blue-tray to reach cube B, right arm places cube on red-tray to create enough space on table, (e) and (f) left arm picks cubes from blue trays and places them on the table, while right arm picks the required cube and places it on the red-tray in alphabetic order.

thus planner needs to move red blocks out of the way first then pick blue or green blocks. Red objects need to moved multiple times to grant an obstacle-free block pick up. Moving a red block from one place to another, may create occlusion for other blocks.
Readers can have look to figure A.3 to catch the idea of how AND/OR graph is implemented for this problem. The graph has 33 nodes and 45 hyper-arcs . Each graph is for moving a blue or green object from a table and place them on their corresponding table, or in case blue or green objects are occluded, red blocks will be moved out of way and blue or green object is moved to another table.

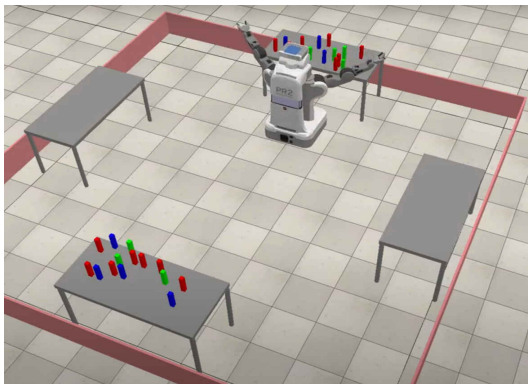
In this benchmark both right and left arms and also the base are involved.

Table 8.5 Statistics of various modules implemented for sorting benchmark

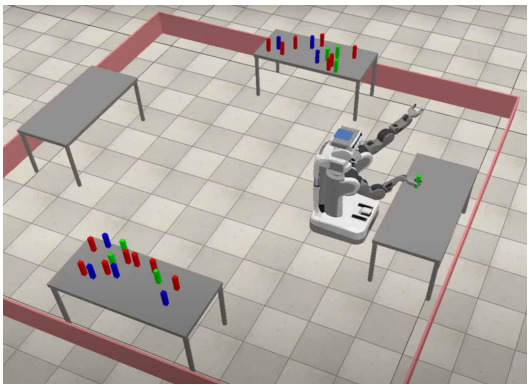
Module	Average	Standard deviation
Depth of AND/OR network	44.5	3.53
AND/OR graph	16.46 [s]	2.61 [s]
Graph search	1.15 [s]	0.123 [s]
Right arm plan	275.07 [s]	27.54 [s]
Left arm plan	325.56 [s]	37.83 [s]
Base plan	93.8[s]	4.65 [s]
Right arm plan attempt	205.11 [s]	16.89 [s]
Left arm plan attempt	279.45 [s]	100.37 [s]
Base plan attempt	90.18 [s]	3.07 [s]

As table 8.5 suggests, on average 44.5 graphs are required to move 7 blue and 7 green objects to their corresponding table, while optimal number of graphs should be 14 according to the graph we implemented. This huge difference roots in red obstacle cubes, where their movements from one place on table to another, in many cases, doesn't make any difference. Planning time and attempt time for both left and right arm seems to be approximately the same, implying their equal contribution in sorting tasks. Figure 8.4, summarizes the whole process steps.

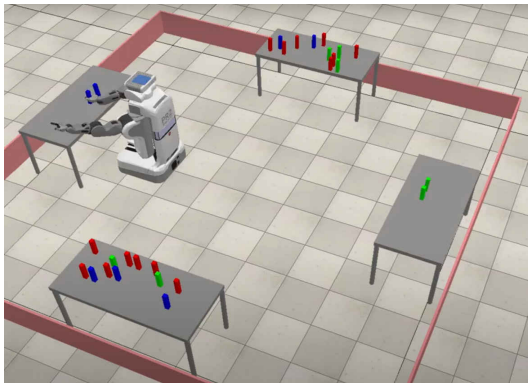
As shown in 8.4a, pr2 robot following it's AND/OR graph, moves to the table with more number of blue and green blocks(in case of equal cubes in both tables, first table is selected) , once on table, closest block to robot is selected to move, in the figure robot picks a green block and in 8.4b, the block is placed on it's corresponding table. This procedure goes on and depicted in figures 8.4c and 8.4d, where 2 blue and 5 green blocks are placed accordingly. Up to this point, red blocks where not occluding other blocks, however in next step, as in figures 8.4e, 8.4f and 8.4g, robot moves away 2 red blocks to reach to a green block in 8.4h.



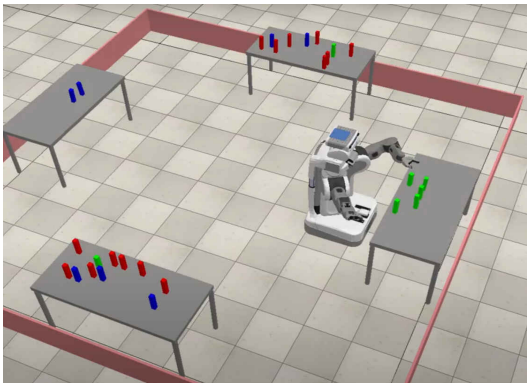
(a)



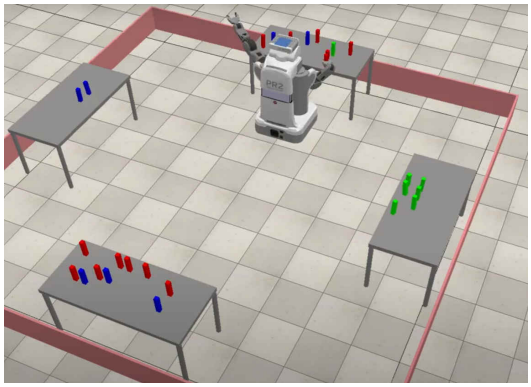
(b)



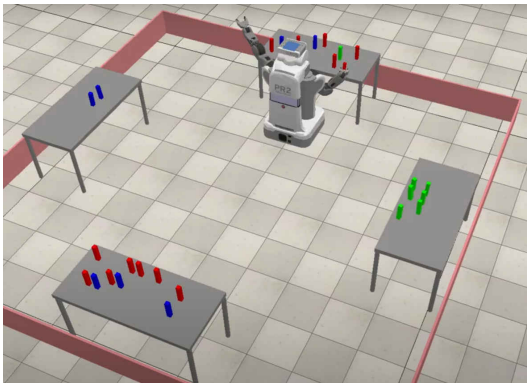
(c)



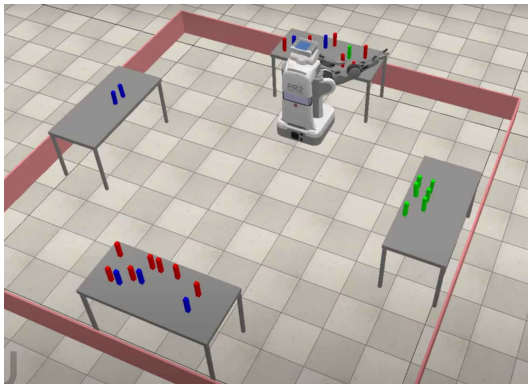
(d)



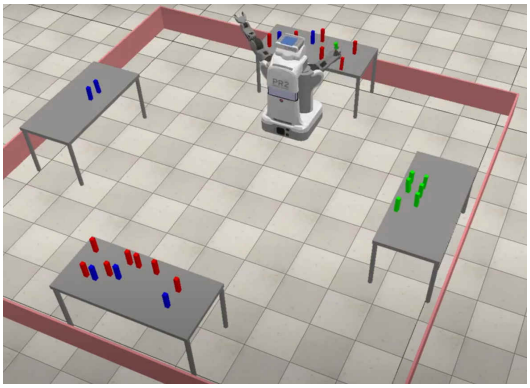
(e)



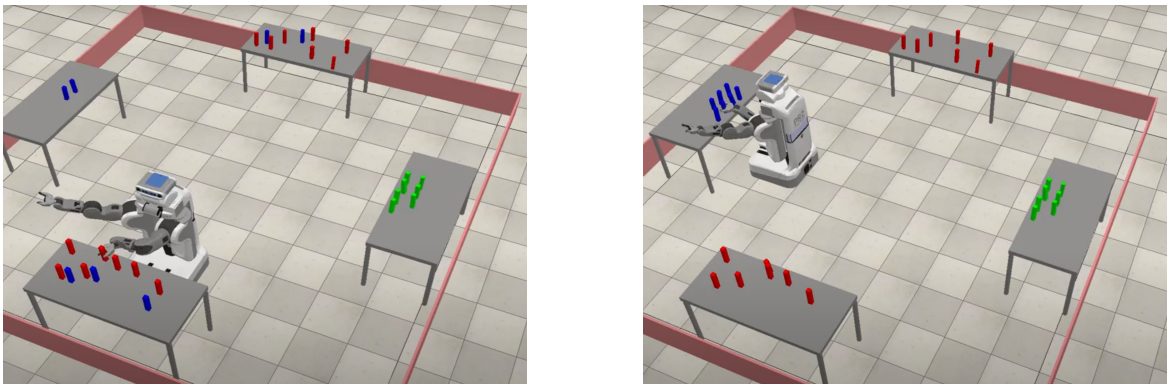
(f)



(g)



(h)



(i) (j)

Figure 8.4 Illustration of sorting object benchmark in simulation environment

The same scenario goes on for blue block on the other table as shown in 8.4i. Eventually as shown in 8.4j, all blue and green blocks are sorted and placed to their tables.

8.4 Non-Monotonic

This problem targets *infeasible task actions* and *non-monotononicity* criteria.

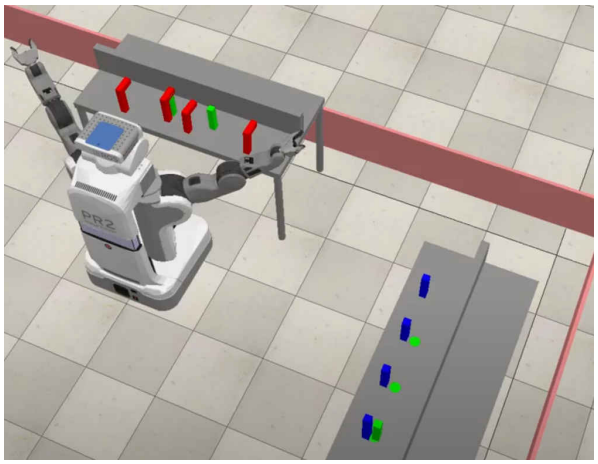


Figure 8.5 PR2 robot in simulation environment solving non-monotonic problem

The objective of this problem is to move all green blocks, as shown in figure 8.5 from the left table to the right table and placing them in their spotted position. Green blocks at left table are occluded by red blocks, so robot arms, first need to move them to create an obstacle-free pick up for green blocks. successively green blocks need to temporarily placed

Table 8.6 Statistics of various modules implemented for non-monotonic benchmark

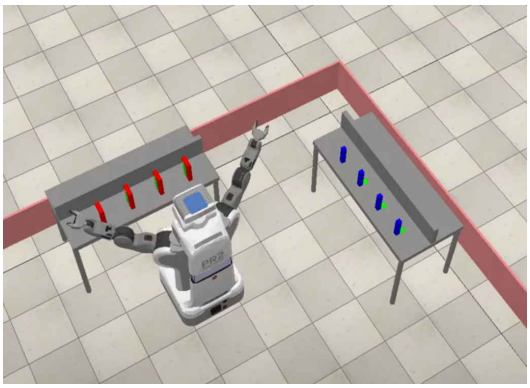
Module	Average	Standard deviation
Depth of AND/OR network	5.5	0.7
AND/OR graph	1.73 [s]	0.07 [s]
Graph search	0.38 [s]	0.18 [s]
Right arm plan	192.36 [s]	17.29[s]
Left arm plan	276.71 [s]	59.38 [s]
Base plan	18.08[s]	3.08 [s]
Right arm plan attempt	131.61 [s]	5.76 [s]
Left arm plan attempt	208.92 [s]	44.74 [s]
Base plan attempt	16.83 [s]	3.57 [s]

on left table, so red blocks can be placed again to their initial positions.

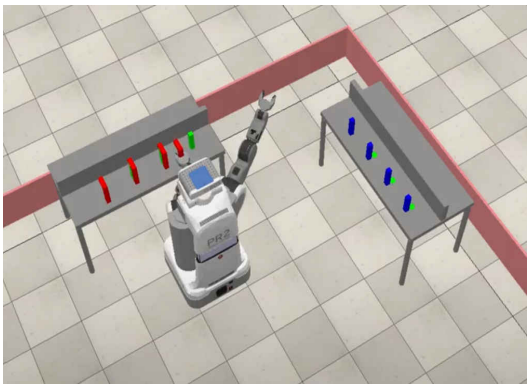
The same applies for blue blocks. They need to me moved to a temporarily place, and once green blocks are placed on their corresponding spot, blue blocks also have to be placed to their initial positions. This challenge excites the *non-monotonicity* criteria in particular.

Totally, there are four green blocks to displace. The AND/OR graph used to implement this problem is shown in figure A.4. It has 34 nodes and 40 hyper-arcs.

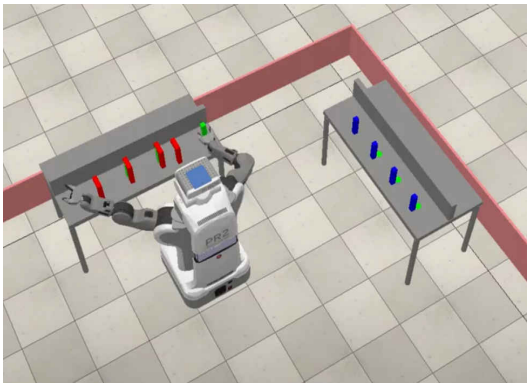
Figure 8.6 depicts the overall process of this problem solution. Once the process starts, there are four green blocks on the left table that are occluded by four red blocks and on the right table there are four blue blocks and four highlighted green spots as shown in 8.6a. Robot approaches to the left table, picks the first red block and places it on the left or right side of the green block as in 8.6b. The side is chosen at run-time depending on motion planner results. Once green block is reachable, it is also picked and placed in the opposite side of the red block, then red block is placed back to it's initial configuration as shown in 8.6c and 8.6d. Right after, green object is picked and transported to the right table as in figure 8.6e and placed either on left or right side of the blue block as in 8.6f, again decision is made at run-time relying on motion planner simulation results. Subsequently blue block is placed opposite to green block placement side as shown in figures 8.6g and 8.6h. Last step is to place respectively blue and green blocks onto their designated places 8.6i. This procedure is kept for other blocks as well. At the end of the scenario all the green blocks should be placed on green spots on the right table, while all red and blue blocks are at their initial configurations as shown in 8.6a.



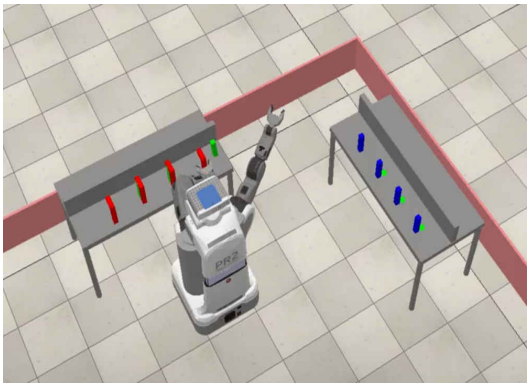
(a)



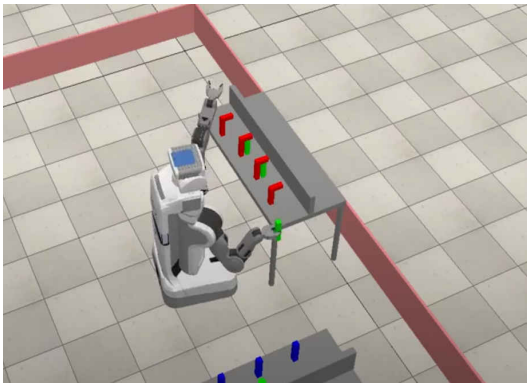
(b)



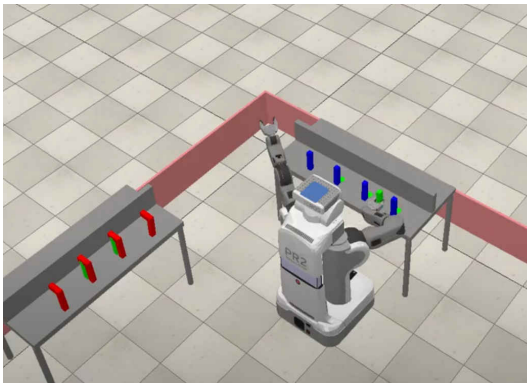
(c)



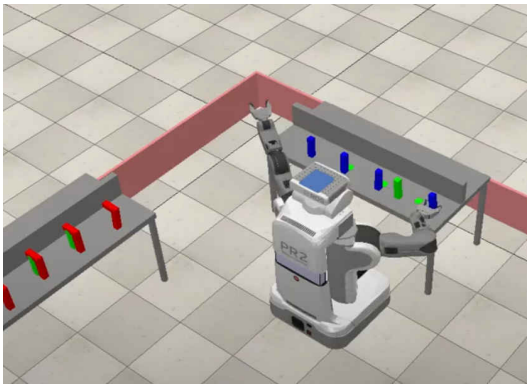
(d)



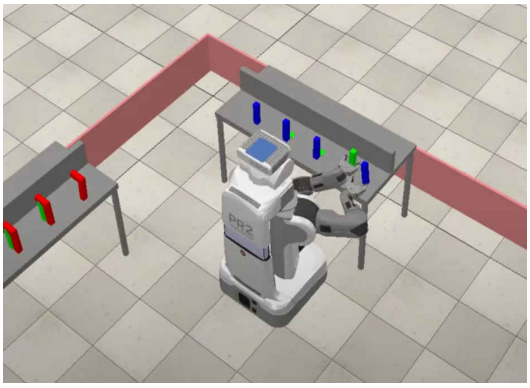
(e)



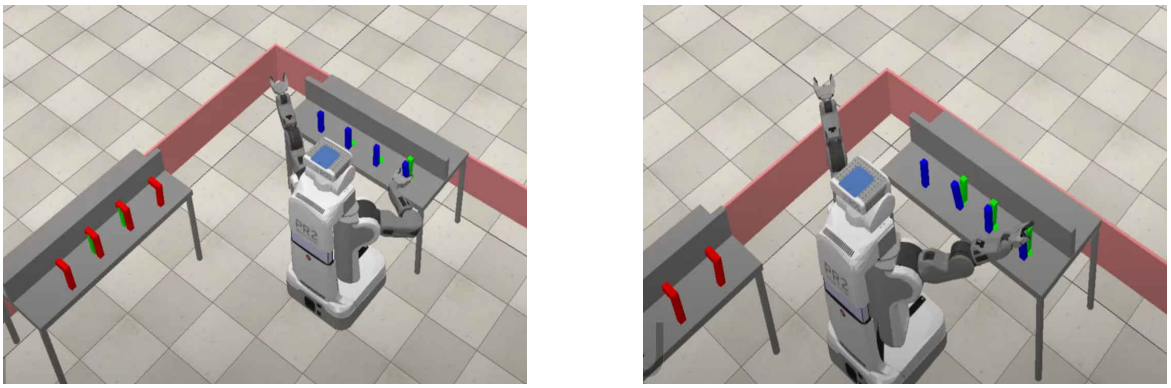
(f)



(g)



(h)



(i) (j)

Figure 8.6 Illustration of various steps of non-monotonic benchmark implementation in simulation environment

Here, to keep a tidy and ordered table, we, right before moving to another table displace blocks to their initial configurations, one can follow alternative paths, such that tidiness principal is met at the end of the scenario, rather than at each inter table movement step.

8.5 Kitchen

Kitchen benchmark excites *infeasible task actions*, *non-monotonicity* and also *non-geometric actions*. Objective of this problem is to *prepare a meal*. It is done by cooking two cabbages, green blocks, washing two glasses, blue blocks and setting the table. There are also four radishes on the shelf as shown in figure 8.7, occluding and obstructing cabbages that need to be moved away. However to keep the kitchen clean and in order, radishes have to be placed to their initial positions.
Note that to wash the blocks, they should be placed on the dishwasher and to cook, they should be placed on the microwave, and to cook blocks, they should be washed before.

The AND/OR graph implementation for this benchmark is quite straight forward, see figure A.5 All the steps need to be followed sequentially and there is no complexity for that. It is made of 48 nodes and 60 hyper-arcs. Statistical data of carious modules for this benchmark are listed in table 8.7.

We describe the overall procedure of kitchen benchmark following the figure 8.8.

Once the process starts, robot moves to the shelf as in figure 8.8a, and picks the obstructing radish and places it somewhere on the shelf, see figure 8.8b. Then robot arm, picks the cabbage and places it in a pre-defined position 8.8c. To keep the shelf tidy, places back the radish to it's initial position 8.8d. Before cooking the cabbage, it is placed on the dishwasher

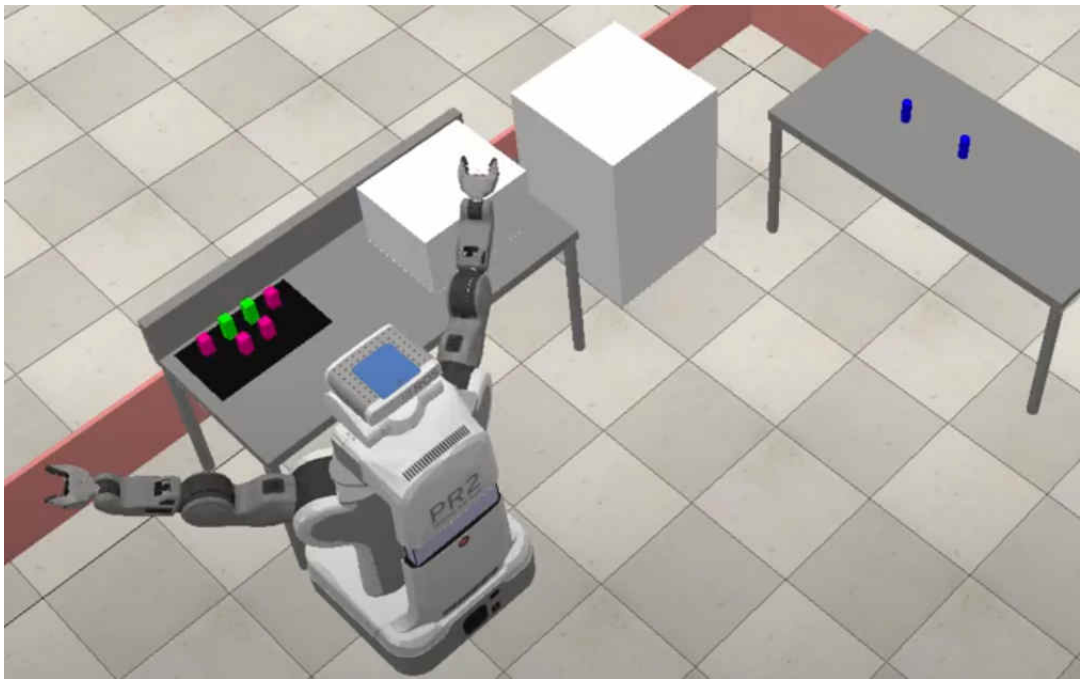
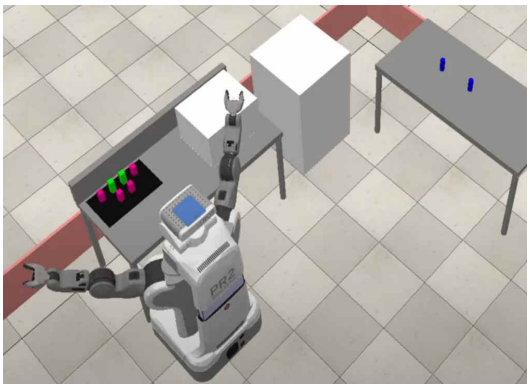


Figure 8.7 Illustration of kitchen benchmark in simulation environment

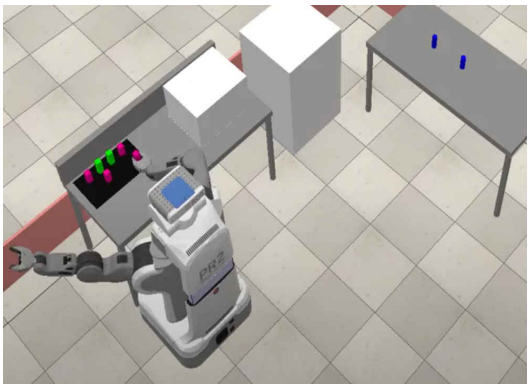
Table 8.7 Statistics of various modules implemented for kitchen benchmark

Module	Average	Standard deviation
Depth of AND/OR network	13.5	4.95
AND/OR graph	7.78 [s]	0.47 [s]
Graph search	0.47 [s]	0.006 [s]
Right arm plan	317.89 [s]	9.52[s]
Left arm plan	121.76 [s]	20.95 [s]
Base plan	56.86[s]	0.017 [s]
Right arm plan attempt	284.06 [s]	29.07 [s]
Left arm plan attempt	84.11 [s]	12.76 [s]
Base plan attempt	57.83 [s]	1.62 [s]

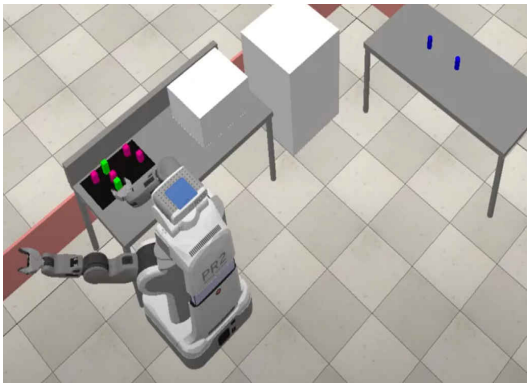
to be cleaned 8.8e, once cabbage is placed on the dishwasher, robot waits for a while (to animate the wait action, robots moves around comes back 8.8f). Then cabbage is placed on the microwave for cooking 8.8g, 8.8h. In the meantime, while cabbage is being cooked, glasses are washed as shown in 8.8i and 8.8j. Now the cabbage is ready!!, it is picked from microwave in 8.8k and placed on the table in 8.8l. Glass is also washed now and is clean enough, thus picked and placed beside the meal as shown in 8.8m. This procedure goes on for all the cabbages and glasses and eventually the table is set for two persons as shown in figure 8.8n. Buon appetito !!!



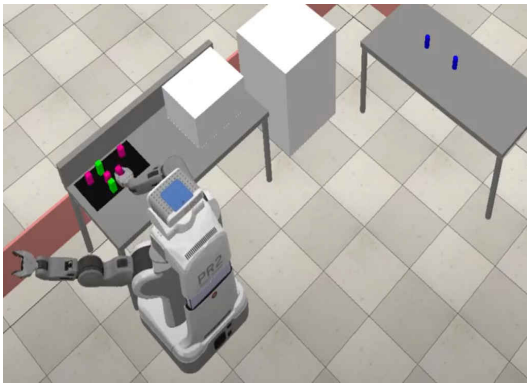
(a)



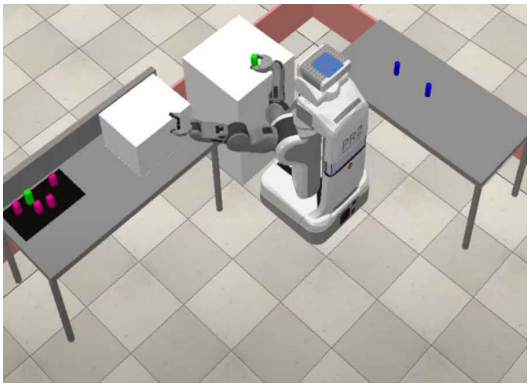
(b)



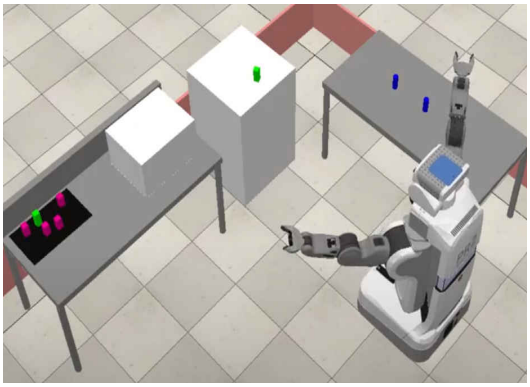
(c)



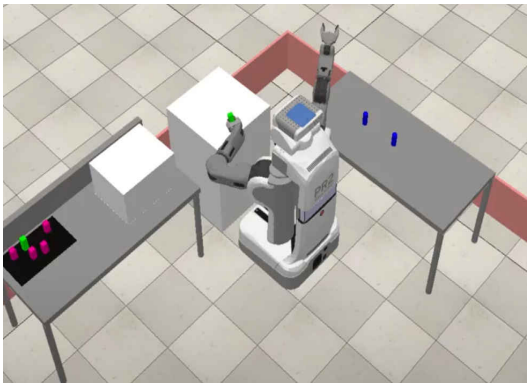
(d)



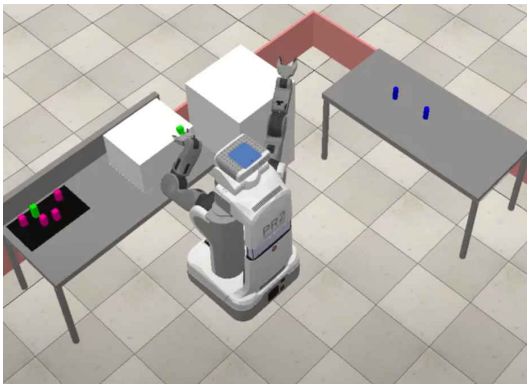
(e)



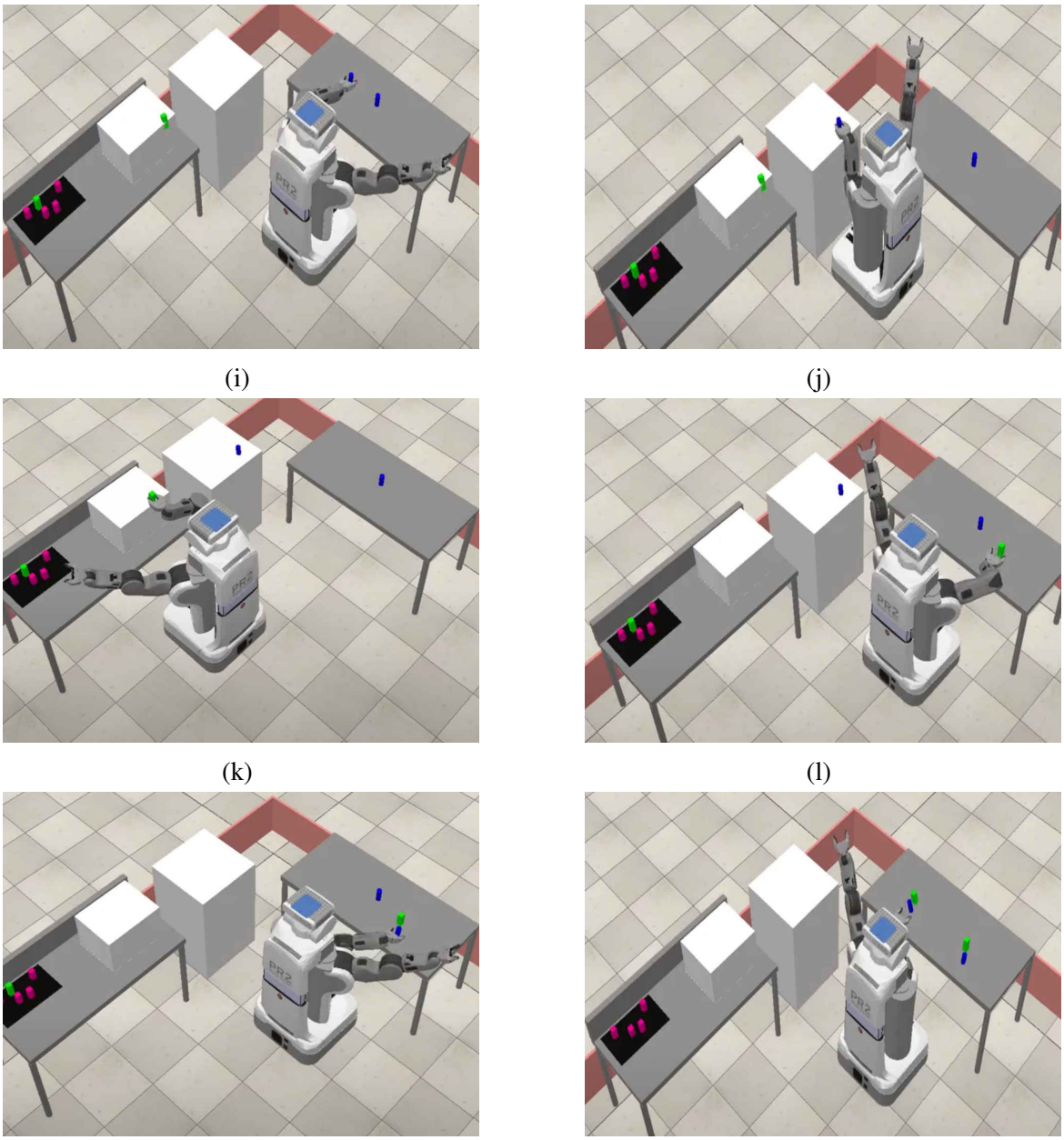
(f)



(g)



(h)



(i) (j)

(k) (l)

(m) (n)

Figure 8.8 Illustration of various steps of kitchen benchmark implementation in simulation environment

Whole the cooking process, takes on average iterating 13.5 AND/OR graphs as table 8.7 shows. It seems like, variation from the average is more than of 30% in different tests. We can speculate that this high variation is coming from failures of motion planning results. As we mentioned previously motion planner is based on random algorithms and it's functionality varies from time to time. Unfortunately this affects our architecture in high order. Moreover we can deduce this from high values required for motion planning time, specifically for right arm 317.89 seconds takes only for motion planning, while it's execution time are only 284.06 seconds. This implies that, the majority of motion planning requests are responded with failure result.

Chapter 9

Conclusions and Future work suggestions

9.1 Conclusion

In this thesis, we presented three variants of AND/OR graphs, namely, (1) *c-layer AND/OR graphs*, (2) *Branched AND/OR graphs*, and (3) *Iteratively deepened AND/OR graphs*. Each of these AND/OR graph variants are meant to model the task representation layer for specific problems. In chapter 2, we briefly introduced AND/OR graphs, and discussed their application procedure and explained how we can adopt them for robotic related problems. Later in chapters refchap:concurent, 5 and 7 we expanded AND/OR graphs structures and reformulated them to from all aforementioned three variants AND/OR graphs.

In chapter 3, we introduced the concept of HRC and designed a HRC scenario for object defect inspection problems. We described an ODI scenario, with one human agent and four robot agents together collaboratively following a common goal for ODI. In chapter 4 we presented CONCHRC framework aimed at forming an integrated system architecture with three main layers as follows: i) representation, ii) perception and iii) action layers. Our major contribution, i.e., *c-layer AND/OR graphs* is embedded in representation layer. To evaluate the overall CONCHRC framework, we implemented and ODI scenario according to what described in 3. The experiment involved a human agent and four robot agents as mobile base robot, a manipulator arm mounted on the mobile base, right and left arms of Baxter robot. On the basis of the experiments we carried out, it is possible to make two different remarks.

The first is related to the robustness associated with the overall process. In spite of such faults as unsuccessful robot grasps, or issues related to false positives or negatives when monitoring the activities carried out by human operators, the inherent flexibility of

CONCHRC allows human operators to intervene and manage these issues. This is even more relevant considering that our current setup does not focus on such a robustness level.
The second is the insight that using parallel instances of AND/OR graph representation layers seems to be more efficient with respect to an equivalent, common, single instance model. We observed that the adoption of CONCHRC reduces the overall idle time considerably.
This is an obvious consequence of the fact that the total time needed for a multi human-robot collaboration process to conclude is determined by the maximum one associated with the longest execution branch in the graph. On the contrary, if the HRC process were implemented as a single, non concurrent, model, then the total time would correspond to the sum of all times associated with single cooperation paths. As an example, in our scenario CONCHRC allows for a total collaboration time equal to 310.19 *s*, whereas an equivalent implementation using FLEXHRC the total collaboration time can be up to 866.94 *s*.

In chapter 5, we presented *Branched AND/OR graphs*, and integrated it to a framework similar to that of CONCHRC, but this time with only left and right arms of Baxter robot in presence of a human operator. *Branched AND/OR graphs* target a flexible and adaptable HRC process, such that a higher level of autonomy and flexibility is given to human operator to intervene not only at pre-defined steps of the process, but also whenever desired, at each step the process to intervene, stop the process and change the planning. We considered two kind of human intervention, one in the context of *kinesthetic teaching* and another in the context of *task repetition*. The results of the experiments proved the efficiency of our approach and showed that *Branched AND/OR graphs* provide more degrees of freedom for human operators in HRC processes and considerably reduces robot failures and cognitive stress on human operators.
In chapter 6, we introduced TAMP problems in robotics and formalised Task and motion relationships. We positioned our approach in the literature and compared it qualitatively with the existing methods.
In chapter 7, we presented *Iteratively deepened AND/OR graphs*, and encapsulated it into the task representation layer of TMP-IDAN framework. We formulated *Iteratively deepened AND/OR graphs* and evaluated their computational complexity.
We evaluated TMP-IDAN in two distinct scenarios, first implemented for single robot single target retrieval from a cluttered top table and second, implemented for multi robot multi target retrieval from a cluttered top table. The results provided from these experiments approved scalability and applicability of out approach for both real-world and simulation environment scenarios effectively. We were satisfied with results of our approach at both qualitative and quantitative aspects compared to current existing methods in the literature.

Later in chapter 8 to prove that TMP-IDAN framework can be generalised to all possible TAMP problem rather than cluttered table problem, we implement this framework for five TAMP benchmarks. These benchmarks include one or more tough criteria for planners as evaluation point. The criteria are as follows: (i) Infeasible task actions, (ii) Large task spaces, (iii) Motion/Task Trade-off, (iv) Non-monotonicity, (v) Non-geometric actions.
We successfully validated our approach by implementing all of these benchmarks and meeting all the mentioned criteria. We showed that AND/OR graphs, in particular *Iteratively deepened AND/OR graphs* are able to model any TAMP problem.

9.2 Future work suggestions

Our suggestions encompasses both HRC and TAMP problems. For HRC, we implemented a single scenario for ODI where products were already manufactured. It might be a good practice to introduce and implement other HRC scenarios such as including *manufacturing process*, where humans and robots collaboratively assist in manufacturing process of a product from it's very initial steps. This is mainly doable only in presence of cobots that the safety of human operators is not endangered. Implementing of various scenarios also can give us a better evaluation of our approaches.
Moreover, we focused on human robot interaction problem only from collaboration point of view. Many other aspects of interaction may be added such as physical human robot interaction, where the physical interaction force among robots, human agents and the environment are needed to be considered, since they affect considerably state of collaboration and they may lead also to safety related issues. Thus, one may extent *task planning* problem to to *task-force* planning problem as a suggestion.
In our frameworks, the presence of human agents was modelled into AND/OR graphs and the collaboration was done through communication means using pre-defined human gesture actions. Robots at real-time were not aware of human agents position in the environment. This can put in danger the safety of human operators. Therefore, we suggest, in perception layer of the framework, to add another module to monitor and track human agents constantly and to recognize a series of human activities and geometric positions, in order to adapt the planner to the situation and act accordingly. Additionally, we adopted only vision and inertial sensory data as means of communication between human and robots. In many occasions outputs of human activity recognition were not functioning as expected. An ideal communication mean, on one hand, should recognise properly human intentions and transmit it with

minimum latency. On the other hand, it should bias human unwanted intentions. To tackle this issue, we suggest to adopt multi sensory human activity recognition algorithms to have better communication model.
Regarding TAMP benchmark problems, our validation is done only in simulation environments. Implementing the benchmarks in real-world scenarios would give a more realistic and credible impression to out approach. Note that, simulated environment helped us to relax many real-world limitations and constraints issues. We assumed to a perfect perception system and ideal hardware and control related problems that in real-world scenarios become a crucial hindrance for implementations.
Since TAMP benchmarks are new to the literature, we only could compare our approach qualitatively with existing methods, in the future, the possibility for quantitative comparison might be doable.

Eventually, we argue that it is possible to have a unique and generic purpose framework to include all variants of AND/OR graphs we presented in this thesis to model a standard and acceptable multi-agent collaboration in a manufacturing work space. Off course this modification is done alongside with other layers such as action and perception to grant a safe and efficient framework.

References

[1] Abbeel, P. and Ng, A. Y. (2004). Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the 21st International Conference on Machine learning*, New York, NY, USA.

[2] Adams, J. A. (2005). Human-robot interaction design: Understanding user needs and requirements. In *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, volume 49, pages 447–451. SAGE Publications Sage CA: Los Angeles, CA.

[3] Akgun, B., Cakmak, M., Jiang, K., and Thomaz, A. L. (2012). Keyframe-based learning from demonstration. *International Journal of Social Robotics*, 4(4):343–355.

[4] Angleraud, A., Houbre, Q., and Pieters, R. (2019). Teaching semantics and skills for human-robot collaboration. *Paladyn, Journal of Behavioral Robotics*, 10(1):318–329.

[5] Basile, F., Caccavale, F., Chiacchio, P., Coppola, J., and Curatella, C. (2012). Task-oriented motion planning for multi-arm robotic systems. *Robotics and Computer-Integrated Manufacturing*, 28(5):569–582.

[6] Bertolucci, R., Capitanelli, A., Dodaro, C., Leone, N., Maratea, M., Mastrogiovanni, F., and Vallati, M. (2021). Manipulation of articulated objects using dual-arm robots via answer set programming. *Theory and Practice of Logic Programming*, 21(3):372–401.

[7] Bruno, B., Mastrogiovanni, F., Saffiotti, A., and Sgorbissa, A. (2014). Using fuzzy logic to enhance classification of human motion primitives. In *International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems*, pages 596–605. Springer.

[8] Bryce, D. and Kambhampati, S. (2007). A tutorial on planning graph based reachability heuristics. *AI Magazine*, 28(1):47–47.

[9] Cambon, S., Alami, R., and Gravot, F. (2009). A hybrid approach to intricate motion, manipulation and task planning. *The International Journal of Robotics Research*, 28(1):104–126.

[10] Cao, Z., Hidalgo Martinez, G., Simon, T., Wei, S., and Sheikh, Y. A. (2019). Openpose: Realtime multi-person 2d pose estimation using part affinity fields. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 43(1):172–186.

[11] Capitanelli, A., Maratea, M., Mastrogiovanni, F., and Vallati, M. (2018). On the manipulation of articulated objects in human–robot cooperation scenarios. *Robotics and Autonomous Systems*, 109:139–155.

[12] Chang, C.-L. and Slagle, J. R. (1971). An admissible and optimal algorithm for searching AND/OR graphs. *Artificial Intelligence*, 2(2):117–128.

[13] Chen, F., Sekiyama, K., Cannella, F., and Fukuda, T. (2013). Optimal subtask allocation for human and robot collaboration within hybrid assembly system. *IEEE Transactions on Automation Science and Engineering*, 11(4):1065–1075.

[14] Cho, K. H., Kim, H. M., Jin, Y. H., Liu, F., Moon, H., Koo, J. C., and Choi, H. R. (2013). Inspection robot for hanger cable of suspension bridge: Mechanism design and analysis. *IEEE/ASME Transactions on mechatronics*, 18(6):1665–1674.

[15] Claes, D. and Tuyls, K. (2014). Human robot-team interaction. In *Artificial Life and Intelligent Agents Symposium*, pages 61–72. Springer.

[16] Crandall, J. W., Oudah, M., Ishowo-Oloko, F., Abdallah, S., Bonnefon, J.-F., Cebrian, M., Shariff, A., Goodrich, M. A., Rahwan, I., et al. (2018). Cooperating with machines. *Nature communications*, 9(1):1–12.

[17] Dantam, N. and Stilman, M. (2013). The motion grammar: Analysis of a linguistic method for robot control. *IEEE Transactions on Robotics*, 29(3):704–718.

[18] Dantam, N. T., Kingston, Z. K., Chaudhuri, S., and Kavraki, L. E. (2016). Incremental task and motion planning: A constraint-based approach. In *Robotics: Science and Systems*.

[19] Dantam, N. T., Kingston, Z. K., Chaudhuri, S., and Kavraki, L. E. (2018). An Incremental Constraint-Based Framework for Task and Motion Planning. *International Journal of Robotics Research, Special Issue on the 2016 Robotics: Science and Systems Conference*, 37(10):1134–1151.

[20] Darvish, K., Bruno, B., Simetti, E., Mastrogiovanni, F., and Casalino, G. (2018a). Interleaved online task planning, simulation, task allocation and motion control for flexible human-robot cooperation. In *2018 27th IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN)*, pages 58–65. IEEE.

[21] Darvish, K., Simetti, E., Mastrogiovanni, F., and Casalino, G. (2020). A hierarchical architecture for human–robot cooperation processes. *IEEE Transactions on Robotics*, , to appear.

[22] Darvish, K., Wanderlingh, F., Bruno, B., Simetti, E., Mastrogiovanni, F., and Casalino, G. (2018b). Flexible human–robot cooperation models for assisted shop-floor tasks. *Mechatronics*, 51:97–114.

[23] De Mello, L. H. and Sanderson, A. C. (1990). And/or graph representation of assembly plans. *IEEE Transactions on robotics and automation*, 6(2):188–199.

[24] De Moura, L. and Bjørner, N. (2011). Satisfiability modulo theories: introduction and applications. *Communications of the ACM*, 54(9):69–77.

[25] De Santis, A., Siciliano, B., De Luca, A., and Bicchi, A. (2008). An atlas of physical human–robot interaction. *Mechanism and Machine Theory*, 43(3):253–270.

[26] de Silva, L., Pandey, A. K., Gharbi, M., and Alami, R. (2013). Towards combining htn planning and geometric task planning. In *RSS Workshop on Combined Robot Motion Planning and AI Planning for Practical Applications*.

[27] Dornhege, C., Eyerich, P., Keller, T., Trüg, S., Brenner, M., and Nebel, B. (2009a). Semantic Attachments for Domain-Independent Planning Systems. In *International Conference on Automated Planning and Scheduling (ICAPS)*, pages 114–121, Thessaloniki, Greece.

[28] Dornhege, C., Gissler, M., Teschner, M., and Nebel, B. (2009b). Integrating symbolic and geometric planning for mobile manipulation. In *Safety, Security & Rescue Robotics (SSRR), IEEE International Workshop on*, pages 1–6. IEEE.

[29] Erdem, E., Haspalamutgil, K., Palaz, C., Patoglu, V., and Uras, T. (2011). Combining high-level causal reasoning with low-level geometric reasoning and motion planning for robotic manipulation. In *2011 IEEE International Conference on Robotics and Automation*, pages 4575–4581. IEEE.

[30] Esmaeilian, B., Behdad, S., and Wang, B. (2016). The evolution and future of manufacturing: A review. *Journal of Manufacturing Systems*, 39:79–100.

[31] Garrett, C. R., Lozano-Perez, T., and Kaelbling, L. P. (2018a). FFRob: Leveraging symbolic planning for efficient task and motion planning. *The International Journal of Robotics Research*, 37(1):104–136.

[32] Garrett, C. R., Lozano-Perez, T., and Kaelbling, L. P. (2018b). Ffrob: Leveraging symbolic planning for efficient task and motion planning. *The International Journal of Robotics Research*, 37(1):104–136.

[33] Garrett, C. R., Lozano-Pérez, T., and Kaelbling, L. P. (2020a). Pddlstream: Integrating symbolic planners and blackbox samplers via optimistic adaptive planning. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 30, pages 440–448.

[34] Garrett, C. R., Paxton, C., Lozano-Pérez, T., Kaelbling, L. P., and Fox, D. (2019). Online replanning in belief space for partially observable task and motion problems. *arXiv preprint arXiv:1911.04577*.

[35] Garrett, C. R., Paxton, C., Lozano-Pérez, T., Kaelbling, L. P., and Fox, D. (2020b). Online replanning in belief space for partially observable task and motion problems. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5678–5684. IEEE.

[36] Garrett, S. K., Melloy, B. J., and Gramopadhye, A. K. (2001). The effects of per-lot and per-item pacing on inspection performance. *International Journal of Industrial Ergonomics*, 27(5):291–302.

[37] Gerkey, B. P. and Matarić, M. J. (2004). A formal analysis and taxonomy of task allocation in multi-robot systems. *The International Journal of Robotics Research*, 23(9):939–954.

[38] Ghallab, M., Nau, D., and Traverso, P. (2016). *Automated planning and acting*. Cambridge University Press.

[39] Glasauer, S., Huber, M., Basili, P., Knoll, A., and Brandt, T. (2010). Interacting in time and space: Investigating human-human and human-robot joint action. In *19th International Symposium in Robot and Human Interactive Communication*, pages 252–257. IEEE.

[40] Gombolay, M. C., Wilcox, R. J., Diaz, A., Yu, F., and Shah, J. A. (2013). Towards successful coordination of human and robotic work using automated scheduling tools: An initial pilot study. In *Proc. Robotics: Science and Systems (RSS) Human-Robot Collaboration Workshop (HRC)*.

[41] Goodrich, M. A., Schultz, A. C., et al. (2008). Human–robot interaction: a survey. *Foundations and Trends® in Human–Computer Interaction*, 1(3):203–275.

[42] Havur, G., Haspalamutgil, K., Palaz, C., Erdem, E., and Patoglu, V. (2013). A case study on the tower of hanoi challenge: Representation, reasoning and execution. In *2013 IEEE International Conference on Robotics and Automation*, pages 4552–4559. IEEE.

[43] Hawkins, K. P., Bansal, S., Vo, N. N., and Bobick, A. F. (2014). Anticipating human actions for collaboration in the presence of task and sensor uncertainty. In *2014 IEEE international conference on robotics and automation (ICRA)*, pages 2215–2222. IEEE.

[44] He, K., Lahijanian, M., Kavraki, L. E., and Vardi, M. Y. (2015). Towards manipulation planning with temporal logic specifications. In *2015 IEEE international conference on robotics and automation (ICRA)*, pages 346–352. IEEE.

[45] Helmert, M. (2006). *Solving planning tasks in theory and practice*. PhD thesis, Albert-Ludwigs-Universität Freiburg.

[46] Henkel, C., Abbenseth, J., and Toussaintl, M. (2019). An optimal algorithm to solve the combined task allocation and path finding problem. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4140–4146. IEEE.

[47] Huber, M., Knoll, A., Brandt, T., and Glasauer, S. (2010). When to assist?-modelling human behaviour for hybrid assembly systems. In *ISR 2010 (41st International Symposium on Robotics) and ROBOTIK 2010 (6th German Conference on Robotics)*, pages 1–6. VDE.

[48] Johannsmeier, L. and Haddadin, S. (2016). A hierarchical human-robot interaction-planning framework for task allocation in collaborative industrial assembly processes. *IEEE Robotics and Automation Letters*, 2(1):41–48.

[49] Kaelbling, L. P. and Lozano-Pérez, T. (2011). Hierarchical task and motion planning in the now. In *Robotics and Automation (ICRA), IEEE International Conference on*, pages 1470–1477. IEEE.

[50] Kaelbling, L. P. and Lozano-Pérez, T. (2012). Integrated robot task and motion planning in the now. Technical Report 2012-018, Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology.

[51] Kaelbling, L. P. and Lozano-Pérez, T. (2013). Integrated task and motion planning in belief space. *The International Journal of Robotics Research*, 32(9-10):1194–1227.

[52] Karaman, S. and Frazzoli, E. (2011). Sampling-based algorithms for optimal motion planning. *The international journal of robotics research*, 30(7):846–894.

[53] Karami, H., Darvish, K., and Mastrogiovanni, F. (2020a). A Task Allocation Approach for Human-Robot Collaboration in Product Defects Inspection Scenarios. In *29th IEEE International Conference on Robot and Human Interactive Communication*.

[54] Karami, H., Darvish, K., and Mastrogiovanni, F. (2020b). A task allocation approach for human-robot collaboration in product defects inspection scenarios. In *Proceedings of the 29th IEEE International Conference on Robot and Human Interactive Communication (RO-MAN)*, Naples, Italy.

[55] Kattepur, A. (2019). Workflow composition and analysis in industry 4.0 warehouse automation. *IET Collaborative Intelligent Manufacturing*, 1(3):78–89.

[56] Kawaguchi, Y., Yoshida, I., Kurumatani, H., Kikuta, T., and Yamada, Y. (1995). Internal pipe inspection robot. In *Proceedings of 1995 IEEE International Conference on Robotics and Automation*, volume 1, pages 857–862. IEEE.

[57] Kock, S., Vittor, T., Matthias, B., Jerregard, H., Källman, M., Lundberg, I., Mellander, R., and Hedelind, M. (2011). Robot concept for scalable, flexible assembly automation: A technology study on a harmless dual-armed robot. In *2011 IEEE International Symposium on Assembly and Manufacturing (ISAM)*, pages 1–5. IEEE.

[58] Korsah, G. A., Stentz, A., and Dias, M. B. (2013). A comprehensive taxonomy for multi-robot task allocation. *The International Journal of Robotics Research*, 32(12):1495–1512.

[59] Kuffner, J. J. and LaValle, S. M. (2000). Rrt-connect: An efficient approach to single-query path planning. In *Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on*, volume 2, pages 995–1001. IEEE.

[60] Lagriffoul, F., Dantam, N. T., Garrett, C., Akbari, A., Srivastava, S., and Kavraki, L. E. (2018a). Platform-independent benchmarks for task and motion planning. *Robotics and Automation Letters*.

[61] Lagriffoul, F., Dantam, N. T., Garrett, C., Akbari, A., Srivastava, S., and Kavraki, L. E. (2018b). Platform-independent benchmarks for task and motion planning. *IEEE Robotics and Automation Letters*, 3(4):3765–3772.

[62] Lasota, P. A., Fong, T., Shah, J. A., et al. (2017). A survey of methods for safe human-robot interaction. *Foundations and Trends® in Robotics*, 5(4):261–349.

[63] Latombe, J.-C. (1991). *Robot Motion Planning*. Kluwer Academic Publishers.

[64] Lemaignan, S., Warnier, M., Sisbot, E. A., Clodic, A., and Alami, R. (2017). Artificial cognition for social human–robot interaction: An implementation. *Artificial Intelligence*, 247:45–69.

[65] Levine, S. J. and Williams, B. C. (2014). Concurrent plan recognition and execution for human-robot teams. In *Twenty-Fourth International Conference on Automated Planning and Scheduling*.

[66] Malik, A. A. and Bilberg, A. (2019). Complexity-based task allocation in human-robot collaborative assembly. *Industrial Robot: The International Journal of Robotics Research and Application*, 46(4):471–480.

[67] McDermott, D., Ghallab, M., Howe, A., Knoblock, C., Ram, A., Veloso, M., Weld, D., and Wilkins, D. (1998). PDDL- The Planning Domain Definition Language. In *AIPS-98 Planning Competition Committee*.

[68] Meneweger, T., Wurhofer, D., Fuchsberger, V., and Tscheligi, M. (2015). Working together with industrial robots: Experiencing robots in a production environment. In *2015 24th IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN)*, pages 833–838. IEEE.

[69] Mirrazavi Salehian, S. S., Figueroa, N., and Billard, A. (2018). A unified framework for coordinated multi-arm motion planning. *The International Journal of Robotics Research*, 37(10):1205–1232.

[70] Motes, J., Sandström, R., Adams, W., Ogunyale, T., Thomas, S., and Amato, N. M. (2019). Interaction templates for multi-robot systems. *IEEE Robotics and Automation Letters*, 4(3):2926–2933.

[71] Motes, J., Sandström, R., Lee, H., Thomas, S., and Amato, N. M. (2020). Multi-robot task and motion planning with subtask dependencies. *IEEE Robotics and Automation Letters*, 5(2):3338–3345.

[72] Oh, J.-K., Jang, G., Oh, S., Lee, J. H., Yi, B.-J., Moon, Y. S., Lee, J. S., and Choi, Y. (2009). Bridge inspection robot system with machine vision. *Automation in Construction*, 18(7):929–941.

[73] Pandey, A. K., Saut, J.-P., Sidobre, D., and Alami, R. (2012). Towards planning human-robot interactive manipulation tasks: Task dependent and human oriented autonomous selection of grasp and placement. In *2012 4th IEEE RAS & EMBS International Conference on Biomedical Robotics and Biomechatronics (BioRob)*, pages 1371–1376. IEEE.

[74] Preda, N., Manurung, A., Lambercy, O., Gassert, R., and Bonfè, M. (2015). Motion planning for a multi-arm surgical robot using both sampling-based algorithms and motion primitives. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1422–1427. IEEE.

[75] Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., Wheeler, R., and Ng, A. Y. (2009). Ros: an open-source robot operating system. In *Workshop on Open Source Software of the International Conference on Robotics and Automation (ICRA)*. Kobe, Japan.

[76] Rodríguez, C. and Suárez, R. (2016). Combining motion planning and task assignment for a dual-arm system. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4238–4243. IEEE.

[77] Rohmer, E., Singh, S. P. N., and Freese, M. (2013). CoppeliaSim (formerly V-REP): a Versatile and Scalable Robot Simulation Framework. In *Proc. of The International Conference on Intelligent Robots and Systems (IROS)*. www.coppeliarobotics.com.

[78] Rozo, L., Jiménez, P., and Torras, C. (2013). A robot learning from demonstration framework to perform force-based manipulation tasks. *Intelligent Service Robotics*, 6(1):33–51.

[79] Sanderson, A., Peshkin, M., and de Mello, L. H. (1988). Task planning for robotic manipulation in space applications. *IEEE transactions on aerospace and electronic systems*, 24(5):619–629.

[80] Shah, J. A. (2011). *Fluid coordination of human-robot teams*. PhD thesis, Massachusetts Institute of Technology.

[81] Sharon, G., Stern, R., Felner, A., and Sturtevant, N. R. (2015). Conflict-based search for optimal multi-agent pathfinding. *Artificial Intelligence*, 219:40–66.

[82] Simetti, E. and Casalino, G. (2015). Whole body control of a dual arm underwater vehicle manipulator system. *Annual Reviews in Control*, 40:191–200.

[83] Smith, D. E. and Weld, D. S. (1999). Temporal planning with mutual exclusion reasoning. In *IJCAI*, volume 99, pages 326–337. Citeseer.

[84] Spence, C. and Zampini, M. (2006). Auditory contributions to multisensory product perception. *Acta Acustica united with Acustica*, 92(6):1009–1025.

[85] Srivastava, S., Fang, E., Riano, L., Chitnis, R., Russell, S., and Abbeel, P. (2014). Combined task and motion planning through an extensible planner-independent interface layer. In *Robotics and Automation (ICRA), IEEE International Conference on*, pages 639–646. IEEE.

[86] Steinfeld, A., Fong, T., Kaber, D., Lewis, M., Scholtz, J., Schultz, A., and Goodrich, M. (2006). Common metrics for human-robot interaction. In *Proceedings of the 1st ACM SIGCHI/SIGART conference on Human-robot interaction*, pages 33–40.

[87] Sucan, I. A. and Chitta, S. (2013). Moveit!

[88] Şucan, I. A., Moll, M., and Kavraki, L. E. (2012). The Open Motion Planning Library. *IEEE Robotics & Automation Magazine*, 19(4):72–82. https://ompl.kavrakilab.org.

[89] Sycara, K. P. (1998). Multiagent systems. *AI magazine*, 19(2):79–79.

[90] Thomas, A., Mastrogiovanni, F., and Baglietto, M. (2020). Towards Multi-Robot Task-Motion Planning for Navigation in Belief Space. In *European Starting AI Researchers' Symposium*. CEUR.

[91] Thomas, A., Mastrogiovanni, F., and Baglietto, M. (2021). MPTP: Motion-planning-aware task planning for navigation in belief space. *Robotics and Autonomous Systems*, 141:103786.

[92] Toussaint, M. (2015). Logic-geometric programming: An optimization-based approach to combined task and motion planning. In *Twenty-Fourth International Joint Conference on Artificial Intelligence*.

[93] Toussaint, M., Munzer, T., Mollard, Y., Wu, L. Y., Vien, N. A., and Lopes, M. (2016). Relational activity processes for modeling concurrent cooperation. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5505–5511. IEEE.

[94] Tsarouchi, P., Michalos, G., Makris, S., Athanasatos, T., Dimoulas, K., and Chryssolouris, G. (2017). On a human–robot workplace design and task allocation system. *International Journal of Computer Integrated Manufacturing*, 30(12):1272–1279.

[95] Umay, I., Fidan, B., and Melek, W. (2019). An integrated task and motion planning technique for multi-robot-systems. In *2019 IEEE International Symposium on Robotic and Sensors Environments (ROSE)*, pages 1–7. IEEE.

[96] Weld, D. S. et al. (2008). Planning with durative actions in stochastic domains. *Journal of Artificial Intelligence Research*, 31:33–82.

[97] Xie, G., Pegn, J., Zhang, X., and Lin, K.-C. (2010). A dynamic task planning based on task subcontracting and dynamic re-decomposition. In *Proceedings of the 29th Chinese Control Conference*, pages 4518–4523. IEEE.

[98] Zlot, R. (2006). An auction-based approach to complex task allocation for multirobot teams, PhD thesis. *Robotics Institute, Carnegie Mellon University, 5000 Forbes Ave.*

Appendix A

AND/OR graphs used for TAMP benchmarks

In this appendix, you can find all the AND/OR graphs we adopted to model TAMP benchmarks.

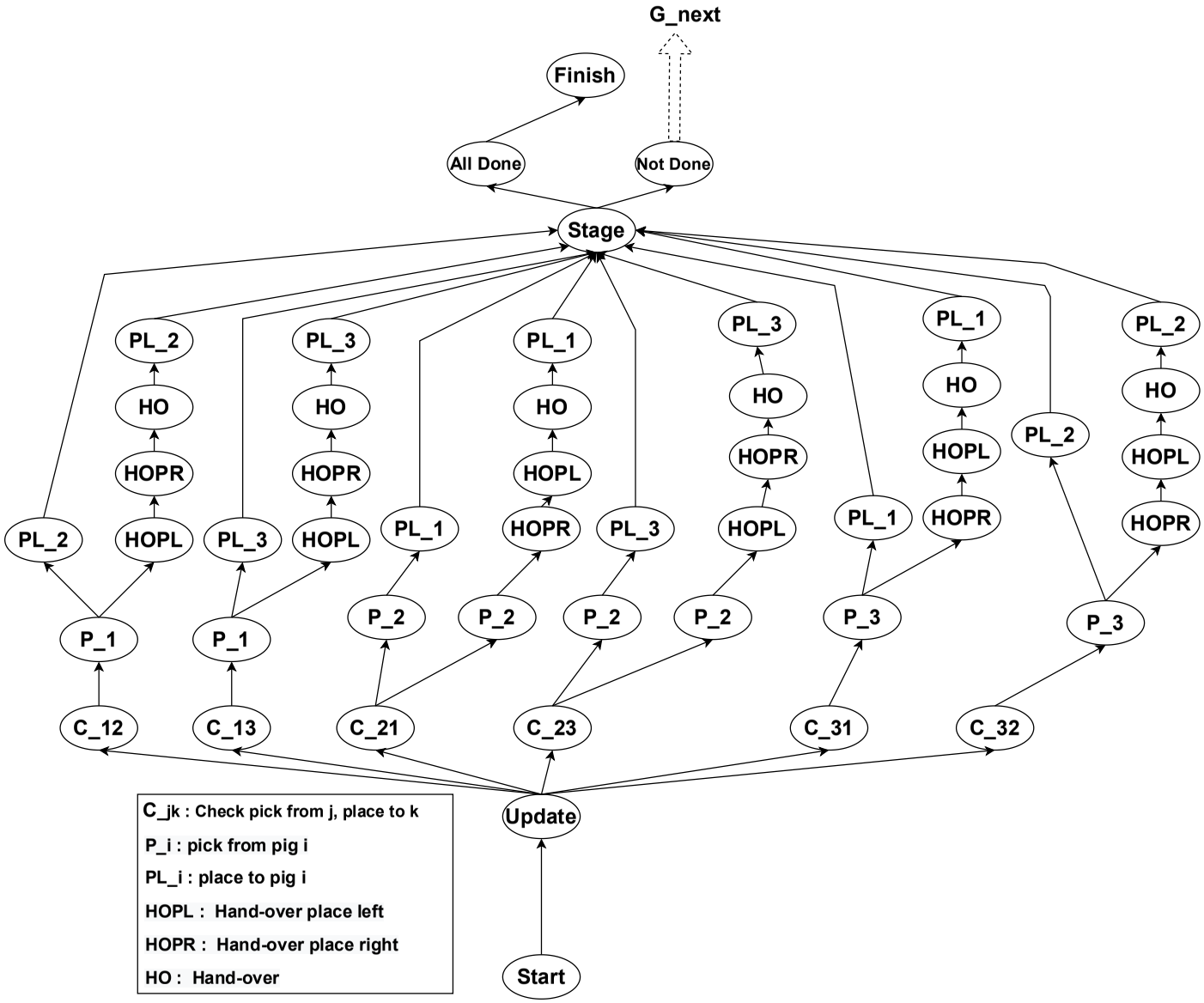


Figure A.1 AND/OR graphs of tower of Hanoi

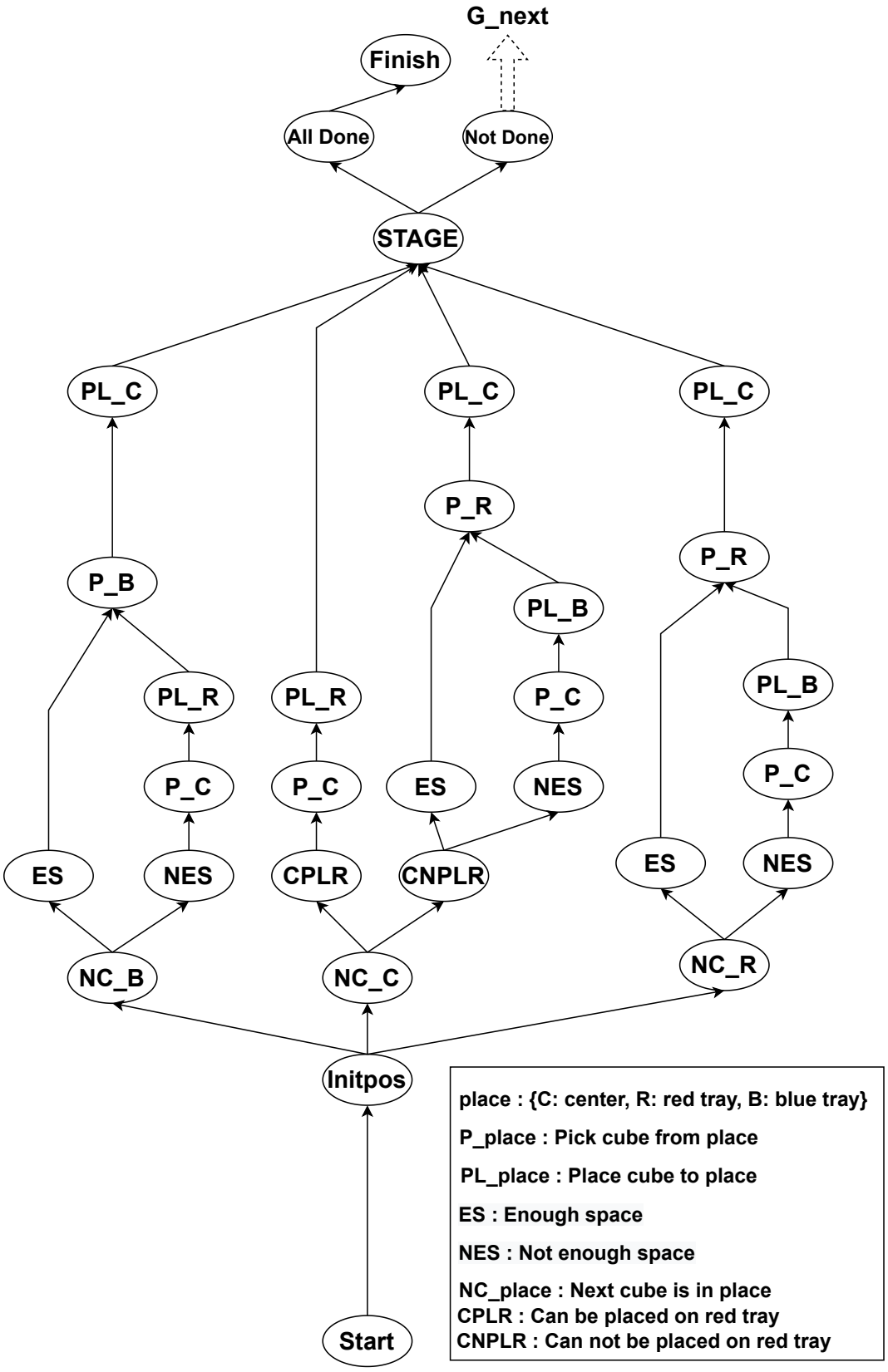


Figure A.2 AND/OR graph of Cube-World 3D

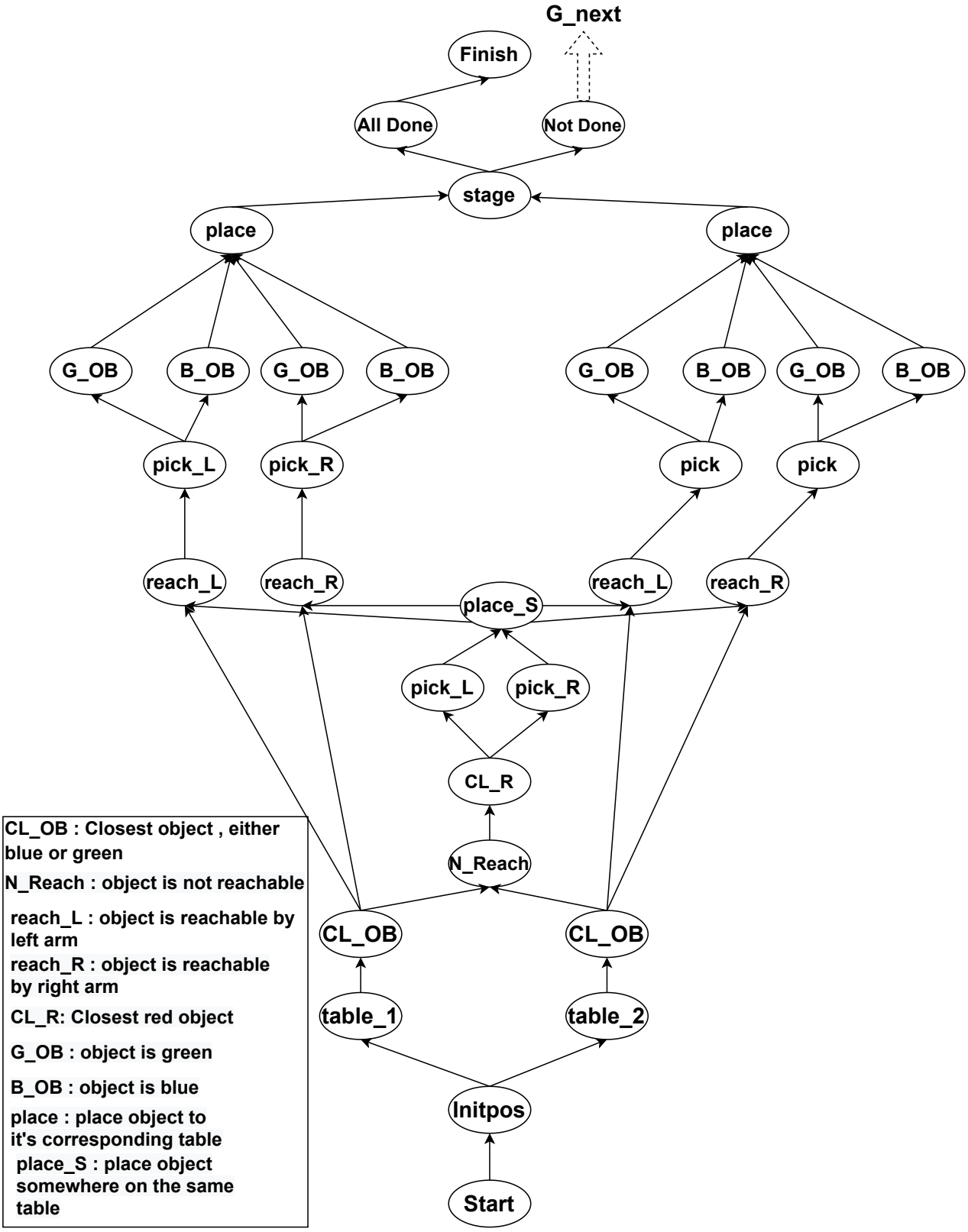


Figure A.3 AND/OR graph used to model Sorting objects benchmark

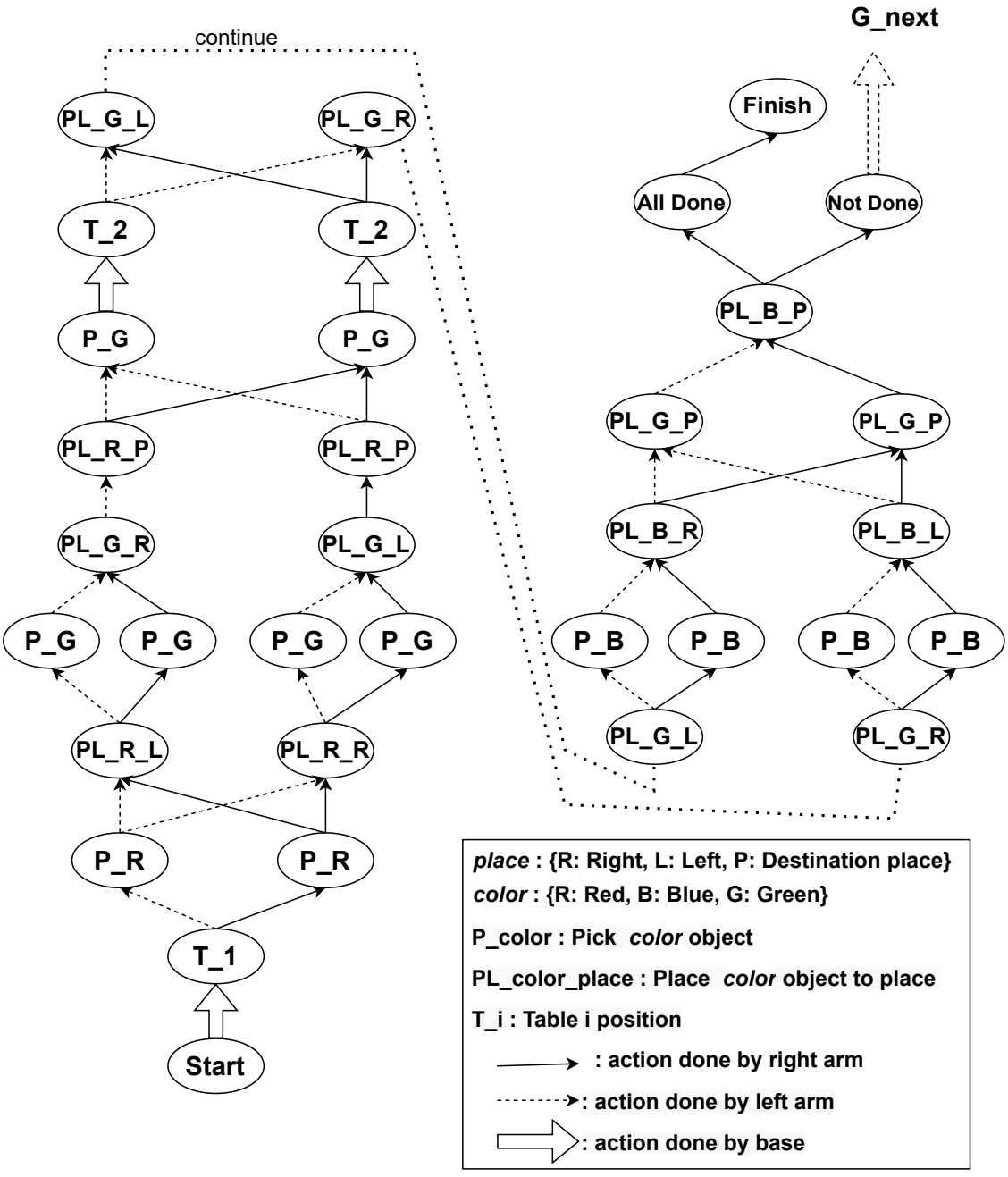


Figure A.4 AND/OR graph used to model Non-monotonic benchmark

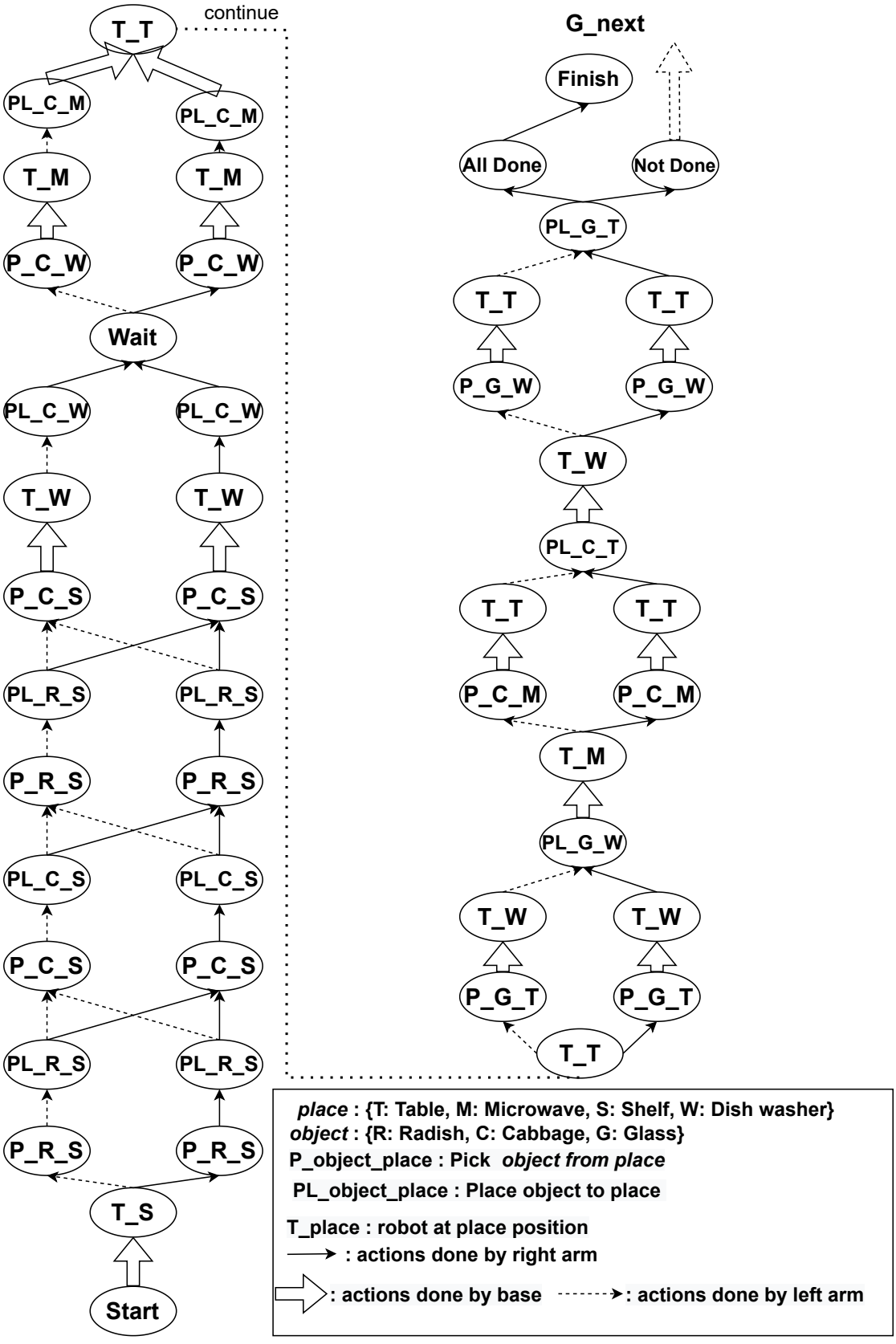


Figure A.5 AND/OR graph used to model Kitchen benchmark