Rheinisch-Westfälische Technische Hochschule Aachen
Knowledge-based Systems Group
Prof. Gerhard Lakemeyer, Ph. D.

Master's Thesis

Centralized Goal Reasoning And Scheduling Using Mixed Integer Programming for Logistics Robots

Mostafa Ashraf Gomaa

Aug 7, 2020

Advisor: Till Hofmann, M. Sc.
Supervisors: Prof. Gerhard Lakemeyer, Ph. D.
Prof. Dr. Matthias Jarke

Contents

Acronyms

BILP Binary Integer Linear Optimization Problems. 11

BS Base Station. 9, 10

CO Combinatorial Optimization. 11

CPA Critical Path Analysis. 16, 17

CPS Cyber-Physical Systems. 6

CS Cap Stations. 10

CX CLIPS Executive. 6, 7, 9, 23, 24, 27, 53

DS Delivery Station. 9

IIoT Industrial Internet of Things. 6

ILP Pure Integer Linear Optimization Problems. 11

IoH Internet of Humans. 6

IoT Internet of Things. 6

LP Linear Programming. 9, 11, 12, 13, 15, 16, 17

MILP Mixed Integer Linear optimization Problem. 11, 12, 27

MIP Mixed Integer Programming. 2, 3, 7, 9, 10, 11, 12, 17, 18, 28, 29, 30, 35, 36, 39, 40, 43, 44, 45, 49, 51, 60

MPS Modular Production Systems. 6, 9, 10, 12, 16, 29, 31, 33, 52

OR Operational Research. 10, 11, 12, 16, 19

PDDL Planning Domain Definition Language. 22, 26, 30, 31, 35, 38, 40, 44, 53

RCLL RoboCup Logistics League. 6, 7, 9, 10, 12, 13, 14, 16, 21, 24, 26, 27, 28, 29, 30, 31, 54, 60

RS Ring Stations. 9

STN Simple Temporal Network. 26

TSP Traveling Salesmen. 11, 15

VRP Vehical Routing Problem. 15

1 Introduction

Industry 4.0 is a term introduced by the German federal government during its 2011 initiative [13] describing what they predicted to be the fourth industrial revolution. Industry 4.0 is characterized by a paradigm shift from centrally controlled to decentralized production processes enabled through the communication between people, machines and resources via Internet of Things (IoT) and Internet of Humans (IoH) [11]. Hermann et al. [11] identified four design principles highlighting industry 4.0 scenarios; interconnection, information transparency, technical assistance and decentralized decisions. A *Smart Factory* is an industry 4.0 application, where decentralized decisions are made by context-aware Cyber-Physical Systems (CPS) interconnected via Industrial Internet of Things (IIoT). Context-aware production agents autonomously determine and pursue short and long term production goals, while utilizing the resources available in their environment.

RoboCup Logistics League (RCLL) is an industrial test bed that replicates a smart factory environment and the production processes therein. Two teams of autonomous robotic agents compete in the production of a dynamically updated list of orders. Robots perform the production by transporting workpieces between a set of team-specific Modular Production Systems (MPS), capable of performing various production operations. Orders arrive online with a variety of production recipes, deadlines and rewards. Since more orders arrive than could be produced, a team's selection strategy of which orders to pursue, as well as its ability to utilize the resources available during production, directly influences the competitiveness of a team (determined by the production score they are able to achieve during game time).

To that end, the Carologistics [1] team (the current world champion of‘ the RCLL) maintains a distributed incremental goal reasoning agent [12]. The agent is implemented using the rule-based CLIPS Executive (CX) [25] that uses a goal reasoning model to guide the program flow via a goal life-cyle. Agents reason locally to come up with production tasks allowed incrementally by the environment. An idle agent assigns itself to the most important task available at the time of reasoning. The importance of a task is determined beforehand by domain experts and coded as static task priorities. Distributed agents reason locally and assign production tasks to themselves, then consequently coordinate the access to the resources required for the task. This is achieved by locking the resources needed for the duration of the task.

[1] https://www.carologistics.org/

Due to its robustness, the current CX agent has been able to achieve remarkable results. Nevertheless, it achieves sub-optimal behavior, which explains why its competitiveness is recently challenged by the advancements in the approaches of other teams. The current distributed approach uses the resources in a greedy fashion, without anticipating their future availability. Moreover, its incremental-reasoning nature lacks coordinating the work done by different agents towards a common deliberated goal. This is bound to waste many opportunities for concurrency allowed by the domain.

This thesis focuses on improving RCLL order production, using a centralized goal-reasoner equipped with a resource scheduler. The scheduler dynamically parses a goal-tree into a corresponding MIP formulation. The MIP model encodes the scheduling problem as a network of production events, and multiple layers of resource flow (i.e., multi commodity flow). The presented MIP formulation is solved to optimality in real-time, minimizing the make-span of production.

This thesis develops a centralized goal-reasoning and scheduling approach, that deliberate a global goal, then schedules the usage of resources by sub-goals, in order to minimize the execution duration of the deliberated goal. The developed approach extends on the incremental goal reasoning agent developed by the Carologistics team. The central goal reasoner formulates a goal tree of tasks, performable by single robots, to produce an order. Causal relations of sub-goals model the precedence of tasks. An integrated scheduler parses the goal-tree and uses a MIP solver to find an optimal solution for the scheduling problem, in real-time. The reasoner supervises the execution of the *scheduled goal-tree* in compliance with a sub-goal execution schedule, and resource allocation schedules (a schedule per resource). Individual actions are dispatch for remote execute by (the allocated) distributed agents. The developed integrated system is a centralized global goal reasoner and resource scheduler that maintains decentralized execution.

A Mixed Integer Programming (MIP) formulation is presented, modeling the complex scenarios found in an RCLL order production. The model encodes a goal tree of production tasks, as a network of (possible) events with commodity requirements and precedence relations. Estimates for traveling and operation durations are incorporated. The objective is to minimize the completion time of the last event (i.e., the makespan or production).

Our approach is evaluated against the incremental agent in a number of simulation runs. It was shown that our approach outperforms the incremental agent when travailing time consumes a big ratio of production time. It also out performs it when complex productions sequence are required. Our approach shows stability to changes in configurations, robot speeds and order complexity. Our executed goal-tree does not break as a result of unexpected schedule violations (due to poor estimates of real world uncertainty).

This thesis is structured into three main sections. Section 2, gives an overview of the theoretical foundations, and the frameworks where the implementation is to be realized. Section 3, covers work related to our approach. Section 4, presents the conceptual model of our approach and elaborates on implementation aspects. Section 4.7, gives a summery and conclusion of our approach.

2 Background

This section gives a detailed overview of the theoretical foundation necessary to understand the approach developed in this thesis. This thesis focus on scheduling the production tasks on an RCLL order. Section 2.1 introduces the RCLL domain and order production.

This thesis develops a *scheduling mode* which dynamically interacts with a MIP solver to generate a MIP formulation of the scheduling problem, and solve it to optimality. Section 2.2 introduces MIP and the solver used by our approach; it demonstrates simple LP models for the two main sub-problems found in *resource scheduling*, the *allocation* (routing) and *sequencing* sub-problems. Section 2.3 covers theoretical aspects of the *Resource Scheduling* problem.

Our *scheduling mode*l is integrated within a centralized goal reasoning model. The goal reasoner initially encodes the production tasks as a goal-tree. A sub-goal is expanded (via a plan library) into different plans. Section 2.4 introduces basic concepts from Automated Planning. Section 2.5, introduces the goal reasoning model and the CLIPS Executive (CX) framework which approach extends.

2.1 RoboCup Logistics League

The RoboCup Logistics League (RCLL) simulates a smart factory scenario with an added element of dual team competition. Two teams of three autonomous robotic agents compete in completing the production duties of a dynamically posted set of orders. An order defines the required quantity and delivery time window of a specific production assembly recipe. Scores are awarded for completed production steps, leading to the delivery of a finished product (RCLL rulebook [3]).

A product is assembled of a *base workpiece*; up to three *colored rings*; a *colored cap.* The difficulty level of producing an order belongs to one of three *complexity* classes. The simplest (C0) is a capped base (no rings required). Orders of higher complexities (C1, C2 and C3) are harder to produce. They require the assembly of a number of rings (indicated by their name).

There are essentially four different MPS types capable of different types of operations. A Base Station (BS) dispenses the base workpieces. A Delivery Station (DS) consumes the finished products and unneeded workpieces. Two Ring Stations (RS), each is responsible for assembling two available ring colors. Two Cap Stations (CS), each is responsible for assembling a cap, buffered at the station at an earlier point.

Even though a workpiece goes through a strict chain of production operations, some MPS's need to be prepared before performing an operation. To mount a cap, a CS is first buffered via dismantling of a cap-carrier (stored at a shelf). Ring assembly requires varying quantities of raw material as payment, prior to the operation; raw materials is be obtained from one of several locations (e.g., BS, CS shelf or any disposable workpiece).

For each team, six team-MPS's are operated by (up to three) autonomous robots. A robot transports a single workpiece at a time, forming a virtual conveyor belt between the MPS's. Moreover, a robot communicates with the MPS's in order to instruct them for performing an operation. RCLL provides a realistic test bed where robotic autonomous agents are responsible for communicating and operating cyber-physical systems in a dynamic environment. A competitive strategy depends in large on a teams ability to utilize the resources during a game, in order to achieve the highest production score. This involves making complex decisions and acting efficiently in a dynamic environment with many uncertainties.

This thesis focuses on minimizing the total duration of order production, by dynamically solving MIP formulation, integrated into a centralized goal reasoning and scheduling model.

2.2 Operational Research

OR (or decision science) is a sub-field of applied mathematics concerned with the application of mathematical methods to the study and analysis of problems involving complex systems [20]. It employs methods (such as mathematical modeling, network analysis and combinatorial optimization) to quantify and optimize the complex-decisions in such systems. Nyor et al. [26] provides a comprehensive yet general introduction to OR.

A *smart factory* is such a complex system, giving rise to several well-studied OR problems. Autonomous agents are required to make complicated decisions about the future and the usage of resources. This thesis presents a MIP formulation which encodes the scheduling problem found in the RCLL domain, in order to improve the real-time task allocation of a team of mobile robots, autonomously operating a smart factory.

Mathematical Optimization OR is often interested in finding the "best possible" decision, restricted by the system's constraints. An *optimization* problem is one of maximizing or minimizing a function of several variables (*Objective function*), subject to equality and inequality constraints and integrality restriction.

Feasible Solutions A feasible solution (or feasible subset) for a mathematical model is a set of values specifying an assignment to the variables of a model, which satisfies the model's constraints. A feasible solution is a subset of the feasible set (represented geometrically as a single point in the feasible region of a model).

Feasible Regions A feasible region (or feasible set) is the set of all points specified by a feasible solution. i.e., it is the set of all feasible subsets.

Optimal Solutions An optimal solution is a feasible solution that gives the best possible value for a desirable objective function.

Combinatorial Optimization (CO) is the problem of finding an optimal solution from a finite set of feasible solutions (the feasible region is finite or countably infinite [29]). Since the feasible set is usually very large, an enumeration and exhaustive search are not viable options. Discrete optimization is considered to consist of integer optimization and CO. Integer optimization is concerned with the efficient allocation of limited resources (which can only be discretely divided) to meet a desired objective. A CO problem is formally defined by a *ground set* ε of objects and a set of associated costs c. The optimal solution is to find an 'allowed' selection of elements from the ground set that has highest (or lowest) additive cost possible. i.e., From a set $\mathcal{F}$ of subsets ε (the feasible set of feasible subsets), the optimal solution is the subset $F \subset \varepsilon$ where $F \in \mathcal{F}$ that yields the maximum (or minimum) value for $c(F) = \sum_{e \in F} c_e$ [23] [14].

Some of the common problems that have combinatorial nature are Knapsack problem, Traveling Salesmen (TSP), and *Resource Scheduling.*

Mixed Integer Programming MIP is the most widely accepted modeling technique in OR, used to formulate the decision processes over several variables, in order to meet a desired objective. Mathematical Programming models aim to maximize or minimize a certain objective function subject to a set of inequality and equality constraints. Linear Programming (LP) is the simplest sub-class to solve, since all variables of a model are continuous. Binary Integer Linear Optimization Problems (BILP) is a model where all variables are binary. Pure Integer Linear Optimization Problems (ILP) occur when the variables are only allowed to have integer values. Mixed Integer Linear optimization Problem (MILP) models contain variables of all the previous types (continuous and discrete).

Objective

$$\max \sum_{j\in B} c_j x_j + \sum_{j\in I} c_j x_j + \sum_{j\in C} c_j x_j$$

Subject to

$$\sum_{j\in B} c_j x_j + \sum_{j\in I} c_j x_j + \sum_{j\in C} c_j x_j \begin{Bmatrix} = \\ \leq \\ \geq \end{Bmatrix} b_j \qquad \forall i \in M \tag{2.1}$$

$$l_j \leq x_j \leq uj \qquad \forall j \in N = B \cup I \cup C \tag{2.2}$$

$$x_j \in \{0,1\} \qquad \forall j \in B \tag{2.3}$$

$$x_j \in \mathbb{Z} \qquad \forall j \in I \tag{2.4}$$

$$x_j \in \mathbb{R} \qquad \forall j \in C \tag{2.5}$$

Nemhauser and Wolsey [23] and L. Hoffman and K. Ralphs [14] provide a general formulation of a MILP problem. B, I and C are the sets of indices for the *binary*, *integer* and *continuous* variables x_j , respectively. Constants l_j and u_j are lower and upper bounds for the values of the variable $x_j, j \in N$. The objective function is a weighted sum where $c_j \in N$ are constants. A set of variable assignments that satisfies (2.1) to (2.5) is is called a *feasible solution.* A feasible solution that yields the maximum (or minimum) possible value to the objective function is called an *optimal solution.*

The problem of optimally scheduling the production sub-goals of an RCLL order, on a set of shared resource (i.e., MPSs and robots) is a combinatorial optimization problem. Combinatorial optimization problems are realized via MIP models, since they requires making discrete decision. The scheduling problem consists of an allocation (routing) sub-problem as well as a sequencing sub-problems. It is convenient to first view each sub-problem independently. Even though, the presented MIP formulation solves the allocation and sequencing sub-problems in unison, each individual sub-problem could be realized vi LP formulation.

2.2.1 Allocation Problem (Routing)

In the RCLL production, the decision of allocating *which* of the identical robots transport a workpiece between *which* MPSs , depends on their expected availability at expected locations. This section gives an intuition into modelling the routing sub-problem, by presenting a simple LP network model.

Commodity Flow Network A *commodity flow network* is often used in OR to model nodes of geographic locations, connected by arcs modeling transportation capacities and unit costs of distributing a commodity (or multiple different commodities) through the underlying network. A commodity can flow through the arcs, with a known unit cost for each, starting at a *source* node that supplies it, satisfying a demand at a *sink* node; possibly passing through transit nodes on several paths. The decision version of the problem is that of planning the flow rate of the *commodity* through each arc; to optimize a certain objective (e.g., minimize total flow cost). The problem can be modeled in terms of ensuring the flow conservation (i.e., *material balance*) at each node.

Material balance (continuity) constraint 1. *A type of constraint often used to represent that the sum total of the quantities going into some process equals the sum total coming out. It takes the form* $\sum_j x_j - \sum_i y_i = 0$.

Minimum Cost Flow It is the problem of finding the flow rate of a *commodity* through each arc of a flow network, satisfying all demand; with an objective of minimizing the additive cost. Edges may have lower or upper bound restrictions on their capacities (Capacitated min. cost flow). Common distribution problems are specializations of this problem. The *transportation problem* considers a (bipartite) network where the commodities flow directly from sources to sinks. In the *transshipment problem*, commodities are allowed to flow through transit sources/sink before arriving to the consumer. The *assignment problem* is a bipartite graph where sources have an availability of single unit and sinks have a requirement of single unit. The *shortest path problem* is reduced to finding the flow of a single unit of commodity from a source to a sink.

Figure 2.1 shows an example of a min. cost flow network abstracted from the RCLL domain. The network depicts the problem of finding the *shortest path* of flow for a *single robot* between the production tasks of a C0 order. The source supplies the robot commodity at a starting location. The sink represents the robot, idling at an end location. Each production task, represented by an intermediate node, has different start and end locations (i.e., the robot picks a workpiece from a start location, releasing it at an end, possibly different, location). The direction and the wight of the arcs convey the traveling duration from an end location of a task to a start location of another. Since the costs are *sequence dependent*, the direction of the arc matters (e.g., finishing task 1 then starting task 4 (from CS_I to CS_O) is different than finishing task 4 then starting task 1 (from DS_I to BS_I).

The problem gives rise to an LP model where each constraint models the *material balance* requirement at a node (constraints 2 and 3) and the objective is to minimize the total cost sum of all edges (equation 1).

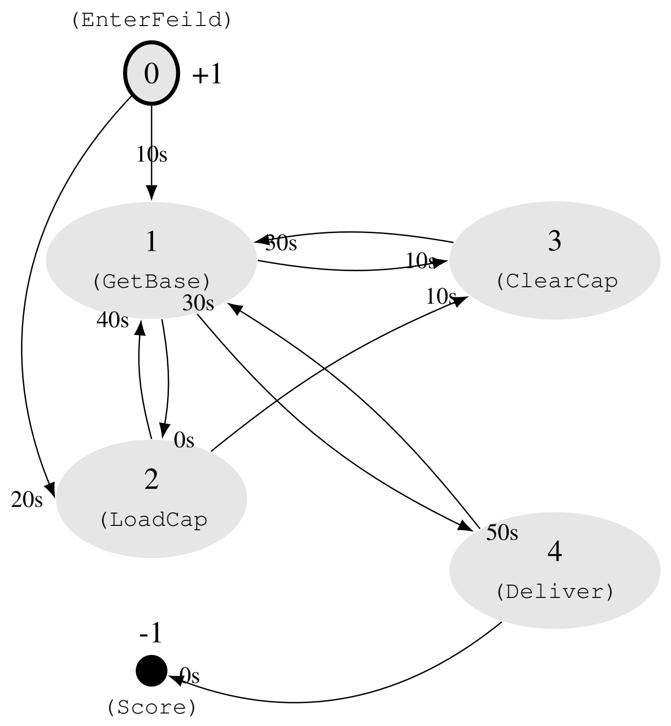


Figure 2.1: Flow network of a single robot commodity, supplied at a source location, through the locations of tasks producing a C0 RCLL order, satisfying the demand at sink location.

Sets

$E = \{1, ..n\}$	Ground set of locations e
C_e	Subset of nodes $i \in E - e$ directly reachable from node $e \in E$
P_e	Subset of nodes $j \in E - e$ that can directly reach node $e \in E$
$S \in E$	Source of the commodity
$Z \in E$	Sink of the commodity

Parameters

$\Delta_{i,j}$	Travailing Duration from node $i \in E$ to node $j \in E$

Decision variables

$X_{i,j}$	The quantity of flow in the edge connecting nodes $i, j \in E$

Objective:

$$\text{Minimize} \sum_i \sum_j \Delta_{ij} X_{ij} \quad i, j \in E \tag{2.6}$$

Constraint:

$$\sum_{i \in C_e} X_{ei} - \sum_{j \in P_e} X_{ej} = 1 \quad e \in \{s, z\} \tag{2.7}$$

$$\sum_{i \in C_e} X_{ei} - \sum_{j \in P_e} X_{ej} = 0 \quad e \in E - \{s, z\} \tag{2.8}$$

This abstract example does not model dependence relations between the tasks; nor that all tasks must be executed. (e.g., getting a base after delivering is an valid path so long as it minimizes the objective). Some of those relations where modeled by only depicting arcs which do not contradict a causal relation. Extending the example to consider the restriction that the robot must visit all tasks, results in the combinatorial Traveling Salesmen (TSP) problem. Extending it further allowing different paths by different robots, results in the combinatorial Vehical Routing Problem (VRP).

2.2.2 Temporal Sequencing

Further considering durations and precedence relations of tasks and their *sequencing* on limited resources, results in different variations of the *scheduling* combinatorial problems. A simpler version of temporally sequencing activities, is to sequence them by only considering their causal relations; which yields a simple LP network model.

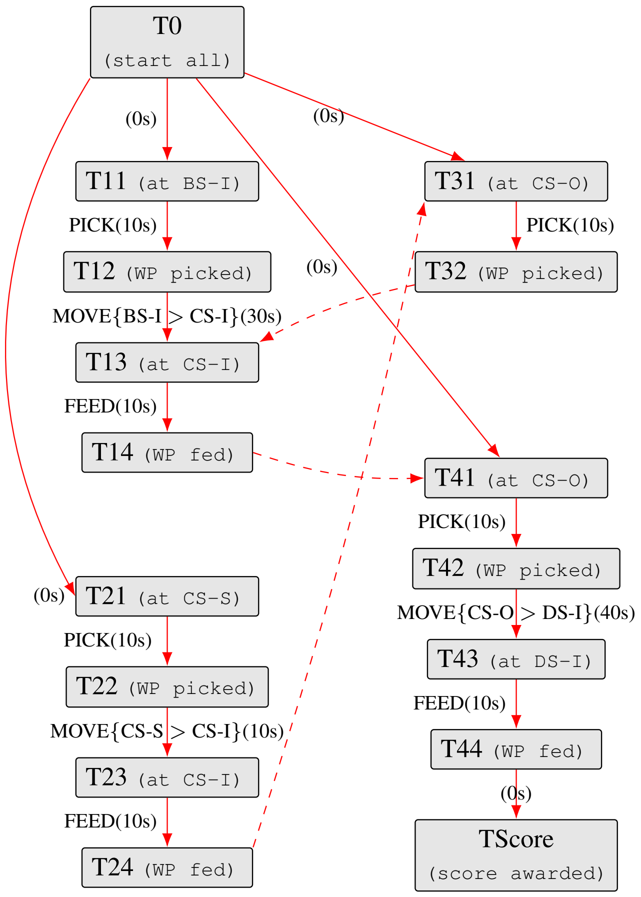


Figure 2.2: Event network of C0 order

Critical Path Analysis (CPA) CPA is a network model used in OR to find the shortest duration needed to execute a group of temporally inter-dependent activities in a project. The analysis answers questions about the temporal restrictions in a project, such as earliest and latest start dates (i.e., slack) for activities. In a network of events, connected by directed weighted *arcs* of activities and temporal precedence, the longest path in the network (i.e., the *critical path*) determines the total duration needed for executing a project. Activities falling on the *critical path* can not be delayed without causing a delay in the project.

The event network in Figure 2.2 depicts a partially ordered plan, of producing a C0 order, in the RCLL. Durative actions are depicted by the weighted arcs connecting event nodes, where the costs are action durations. Dashed arcs model precedence relations between two events (e.g., event $T31$ happens after event $T24$).

The LP formulation of the event network is shown in the following page; where constraint (5) and (6) describe a *sequencing relation* between two events, and the objective (4) is to minimize the completion time of the last event. Solving the model yields an optimal value of 110 seconds for the critical path [$T0$, $T21..T24$, $T31$, $T32$, $T13$, $T14$, $T41..TScore$]. The path is a sequence of inter-dependent events, which collectively consumes the longest execution time. This does not necessarily cover all events. Actions falling on other paths need to be started simultaneously in order to finish the project in the 110 seconds, yet some of them could be delayed without effecting the project (i.e., events with slack).

This example does not account for the limit resource required to perform the activities, such as robots and MPSs. In fact, it assumes that all activities which do not depend on each other, are started simultaneously (by $T0$). It gives an estimate of the production time in a world where resource are unlimited and instantly available. Although impractical, similar estimates could be used in the RCLL to give an initial intuition and a lower bound on the duration needed for producing different orders. A CPA sequences activities by only considering their temporal interdependence, allowing different simultaneous streams of execution. In practical situation, activities can rarely be processed simultaneously as their execution often competes for limited resources. The LP formulation of the CPA follows.

Sets

$E = \{0, .., n\}$	Ground set of events (start of outgoing activity edges)
Parameters	
$\alpha_{i,j}$	Duration of activity on edge ij, where $i, j \in E$
$\gamma_{i,j} \in \{0, 1\}$	If event i precedes event j $\quad (i, j \in E)$
Decision variables	
T_i	Continuous variables representing time of event $(i \in E)$ (the start time for all outgoing activity edges)
T_{Score}	Maximum time taken to finish all activities

Objective: Minimize T_{Score} (2.9)

Constraints:

$$\gamma_{i,j}(-T_i + T_j) \geq \alpha_{i,j} \quad \forall i, j \in E \tag{2.10}$$

$$T_i \geq T_{Score} \quad \forall i \in E \tag{2.11}$$

An important extension to CPA is that of sequencing the activities considering the availability of their *allocated resources*. This extension gives rise to the combinatorial problem of *Resource Scheduling*. Unfortunately, the combinatorial nature of scheduling, requires a great number of discrete decisions. Combinatorial optimization problems could only be realized by a MIP model.

2.2.3 MIP solving

Simplex Algorithm Simplex is a universal algorithm that was been developed by (Dantzig [5]) for solving LP. Based on the observation that the objective function will intersect one of the facets of the polygon of the feasible region. The algorithms starts at looking for *vertices solutions* by evaluating the objective at extreme point of the polygon, then searches through the edges that improve the objective value. For a comprehensive description of the algorithm we refer to [2]

LP Relaxation An LP relaxation of a MIP model, is the LP model obtained by relaxing the integrality restriction of the corresponding MIP model. It is used to give a lower bound estimates on the solution of a MIP model.

Branch And Bound The algorithmic framework was first proposed by Land and Doig [15] as a method to solve disceret programming problems. It remains the most effective paradigm for solving combinatorial optimization problems. The method is based on enumerating the set of feasible solutions of a MIP model, to find the minimum (or maximum) value, by performing state space search on a binary search tree of feasible solutions, employing smart heuristics and pruning techniques. The root, containing the set of all feasible solutions, is recursively expanded into *branches* of disjoint solution subsets. Bounds on the branches are calculated by relaxing the problem (e.g., by solving its LP relaxation); and *fathomed* if either the branch is *bounded* by the value of the best yet found solution (i.e., the bound), or proved *infeasible*. The branching continues till all branches are fathomed. The optimal solution is the best found *integer* solution bound.

Gurobi Solver This section presents common MIP solution strategies and introduces Gurobi, the solver used by our scheduling model.

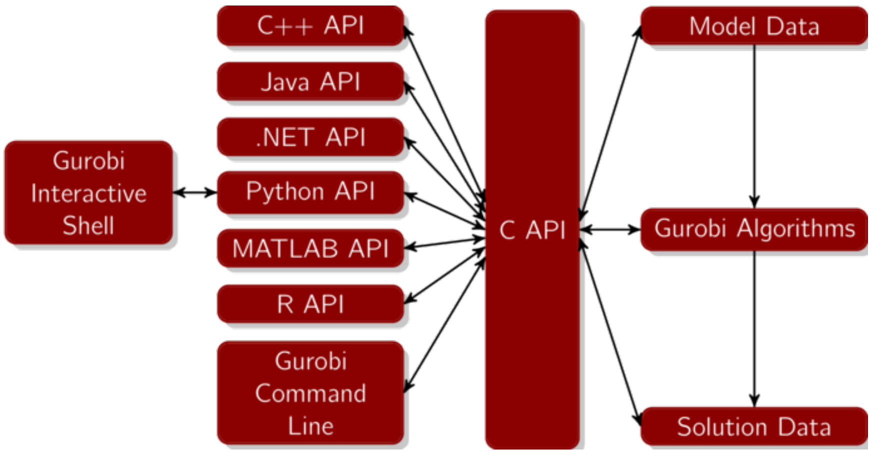


Figure 2.3: Gurobi API [9]

Gurobi [9] optimizer is a high performance commercial solver for mathematical programming. Its capability to exploit parallelism and employ algorithms speedups, put it at the top the standard benchmarks [21] [22]. It provides well-maintained object-oriented interfaces (C++, Java, Python) and comprehensive documentation to technical features.

Gurobi employs several types of cutting plans, pre-solve reduction methods and other advanced algorithms and heuristics to find optimal and near optimal feasible solutions. It supports speedup features such as starting from a feasible solution to guide the search for an optimal solution, which could greatly help reducing solving time. It provides native support for advanced modeling features like Special-Ordered Set constraints, MIN/MAX, ABS, AND/OR and indicator constraints.

2.3 Resource Scheduling

Generally, scheduling refers to the problem of allocating and sequencing the access to a set of resources shared by a set of activities. Instances of the scheduling problem arias in the real world as a need to efficiently utilize limited resource over a time period. In OR, *Machine Scheduling* is the problem of sequencing *operations* achieving a set of *parts* (i.e., jobs) processed by a set of *machines*. The objective is to find a sequencing of operations on each machine, in order to improve an efficiency matrix.

Different sub-problems of scheduling arise depending on the machine environment, jobs characteristics, the nature of precedence relations that exist between operations and the optimization criterion in question. Scheduling classes are studied separately in OR literature to determine their complexity and devise solution strategies. Scheduling literature could be divided into four general classes. The two restrictions shared by most literature, is that a machine is able to process only one operation at a time and that a job can only be processed by a single machine at any given moment.

Single machine scheduling It is a pure sequencing problem where n *uni-operational* jobs (i.e., consisting of a single operation) are processed by a single available machine. It is the simplest scheduling problem possible, where there are only $n!$ possible schedules, determined solely by a job ordering. Generally, a schedule that is solely determined by a common general ordering of jobs, is called a *permutation schedule*.

Parallel machine scheduling A set of uni-operational jobs are processed by a set of *identical, uniform* or *unrelated* machines. This gives rise to the machine allocation sub-problem of scheduling (it requires determining which machines will process which operation). A schedule will involve solving both the machine allocation and sequencing sub-problems simultaneously.

Series machines scheduling (Flow-Shop) This class occurs when machines are arranged in a series forming a fixed pipeline. The problem is of sequencing n jobs for processing by the machine pipeline. Since all jobs are processed by the same machine sequence and an operation can only be processed by a specific machine, the precedence relation between operations of a job is identical for all jobs (starting by the operation on the entry machine and ending by the operation on the exit machine). In the version of the problem, that jobs are not allowed to overtake each other in the pipeline, the problem yields a *permutation schedule*.

Hybrid machine scheduling (Job-Shop) N jobs are processed by m machines. Operations of a job are processed in a known order (specific to each job) on specific machines. Precedence relations between operations and processing durations are explicitly specified.

This general classification abstracts many of the complexities of scheduling found in practical environments. For a more comprehensive classification we refer to the classification of Graham et al. [8], commonly adopted in scheduling literature.

C1 Zero release date: *all jobs are available for processing at time zero*

C2 Sequence independent: *processing times includes the machine setup time, independent of the sequence of processing*

C3 Deterministic: *job and duration information are certain and known in advance*

C4 No breaking: *machines are continuously available*

C5 No preemption *operations can not be interrupted once started*

C6 Uni-operational machines: *machines process at most a single operation at a time*

C7 Single active job's operation: *a job can be processed by at most one machine at a time*

C8 Linear precedence: *relations between operations are arranged in form of single precedence chain*

C9 Explicit machine per operation: *operation can be performed on a specific machine known in advance*

C10 No Job selection: *all jobs need to be executed to completion by the schedule*

Further sub-classes arise when relaxing the restrictive assumptions to a scheduling instance. The $\boldsymbol{\alpha}|\boldsymbol{\beta}|\boldsymbol{\gamma}$ notation of Graham et al. [8] is adopted to help us understand the scheduling problem at hand and its complexity. Graham's survey gives an insight about the complexities of the classes and the reducibility among scheduling problems.

Let $\circ$ denote the empty symbol, a scheduling problem is described by the three field notation $\boldsymbol{\alpha}|\boldsymbol{\beta}|\boldsymbol{\gamma}$.

Machine environment $\alpha = \alpha_1 \ \alpha_2$; $\alpha_1 \in \{\circ, P, Q, R, F, J, O\}$; $\alpha_2 \in \mathbb{Z}^+ \cup \circ$

$\boldsymbol{\alpha_1}$ describes the basic classes indicated earlier. $\circ$ single machine, P identical parallel machines, Q uniform parallel machines (where the processing time is a factor of the machine speed), R unrelated parallel machines, O open shop (same as job shop except the order in which operations are executed is immaterial), F flow-shop, J job-shop.

$\boldsymbol{\alpha_2}$ a positive integer specifies the number of machines as a parameter of the problem instance. $\circ$ means machine number is a variable.

Job characteristics $\beta \subset \{\beta_1, ..., \beta_6\}$

$\boldsymbol{\beta}$ when empty, implies that scheduling instance is in its most basic form. i.e., all of the restrictive assumptions hold.

$\boldsymbol{\beta_1} \in \{pmtn, \circ\}$; *pmtn* means that a job could be interrupted (*preemption*)

$\boldsymbol{\beta_2} \in \{res, res1, \circ\}$; *res*

$\boldsymbol{\beta_3} \in \{prec, tree, \circ\}$; *prec* means linear precedence between jobs could exist, *tree* precedence exists between jobs

$\boldsymbol{\beta_4} \in \{r_j, \circ\}$; r_j means *Release dates* for jobs are specified. If $\beta_4 = \circ$, the release date is zero for all jobs

$\boldsymbol{\beta_5} \in \{m_j \leq \bar{m}, \circ\}$; specifies a constant upper bound on the number of operations in a job is allowed to have (only job-shop)

$\boldsymbol{\beta_6} \in \{1, \underline{p} \leq p_j \leq \bar{p}, \circ\}$; respectively, specifics constant unit processing time or an upper and lower bound on the processing times

Optimality criteria $\gamma \in \{f_{max}, \sum f_i\}$

The only optimization criterion we are interested in here, is minimizing the makespan indicated as C_{max} the completion time of all the jobs. For an overview on the other existing criterion and notation, refer to Graham et al. [8].

The RCLL production scheduling is identified as an extended version of a Job-Shop environment (JSP). In the classical JSP, a part (i.e., job) is produced by a fixed chain-sequence of operations, performed on specified machines. In the RCLL, parts are achievable by alternative operation sequences, on alternative machines, by alternative identical robots. Moreover, operations have in-tree precedence relations (possibly with other parts), and robots have uncertain travel times.

2.4 Planning

In the most abstract sense, *Planning* is the process of deliberating and admitting a sequence of activities, that takes us form a starting state, to a desirable state. In operational research, *Resource Planning* is concerned with scheduling the usage of scares resources between activities to meat a defined efficiency matrix.

In Artificial Intelligence, Ghallab et al. [7] defines *Automated Plan Synthesis* as "Computationally automating the deliberation process of choosing and organizing actions, by anticipating their expected outcomes, to best achieve some objective". Traditionally, the difficulty of the planning problem depends on the aspects we choose to model or simplify. Its best seen in the light of the following 8 restrictive assumption.

A0 Finite: The system has finitely many states, actions, events.

A1 Fully Observable: Complete knowledge over the state is attainable.

A2 Deterministic: Application of an action brings a deterministic system to single state.

A3 Static: State only changes when an action is applied by the controller. (i.e., no exogenous events)

A4 Restricts Goals: The planner handles only restricted goals that are represented as an explicit goal state or a set of goal states. Extended goals like ones seek to optimize a utility are not handled under this restriction.

A5 Sequential Plans: A solution plan is a linearly ordered sequence of actions.

A6 Implicit Time: Actions are instantaneous state transitions with no duration.

A7 Offline Planning: Planner is unaware of any changes that come up after it starts planning. It plans form an initial state to a goal state regardless what comes up.

In this thesis we consider strategies to deal with relaxing assumptions A3, A4, A5, A6,A7

2.4.1 Classical and Non-Classical Planning

Classical planning is planning under the restrictive model (i.e., an environment where all the restrictive assumption 0 to 7 are met). In this context, planning is reduced to finding a successful sequence of actions, that transforms the state of the world from an initial state, to a goal state. That sequence of actions is called a *Sequential Plan*. A world model is represented by a set of true propositions under the closed world assumption (meaning that only proposition that hold are considered true). Actions are state transition operators. An action is only applicable if a set of propositions, called preconditions, holds in the world model. Applying an action changes the state of the world by adding and/or deleting a set propositions, those are called effects of an action. The complexity of classical planning is in essence the complexity of the search algorithm used to find a plan. Since the tree of possible states could get astronomically large, a fair amount of research has been dedicated in finding smart methods and heuristic in order to generate a plan in a reasonable time.

2.4.2 Domain Definition Language (PDDL)

PDDL was introduced by McDermott [19] as an effort to create a unified standardized language for representation and exchange of planning domain models. It models STRIPS fashion actions, as instantaneous state transitions from a state where some preconditions are satisfied, into another where the post-conditions are applied. The syntax is Lisp-like parenthesized. It creates separate models for describing the behavioral capabilities of the system (possible actions) than for describing the characteristics of the system (like initial state and goals). This corresponds to a domain model and a problem model separated in different files. One of the limitations of PDDL is that is can only model predicates symbolically.

PDDL2.1 Due the limited expressiveness of PDDL restricting it to model classical planning problems, Fox and Long [6] proposed an extension to the language. The extension allows the representation numeric fluents, durative actions, conditional effects and plan metric.

2.5 Goal Reasoning Model

Agents following a goal reasoning model are able to deliberate on and self-select their objective (Aha [1]). They are able to reason about which goals to pursue in different situations, rather than only achieving fixed objectives. In this section we present the CLIPS Executive (CX) integrated goal reasoning system [25] , where the proposed approach is to be implemented.

2.5.1 CLIPS Executive

The CLIPS Executive (CX) [25] is an integrated goal reasoning and executive system, implemented using the CLIPS rule-based production system [31], as an effort to bridge the gap between plan synthesis and plan execution by interleaving them into a single program flow. The program flow uses an explicit representation of goals (Aha [1]) adhering to a specific goal life cycle (Roberts et al. [28]). The system achieves robustness in execution by allowing continues monitoring and reasoning about the states of goals and actions, allowing it to influence the flow of execution.

Goal A *goal* is the core data structure modeling a constraint or an objective that an agent is interested in achieving or maintaining. In CX, a *goal type* describes whether the goal is intended for *achieving* or *maintaining* a certain condition. Goals arise either as a result of situational reasoning about the environment, as a result of execution of another goal or by an external influence (e.g., commanded by an authorized master agent). A goal is a grounded instance of a *goal class*, defining a category of goals.

Goal Life Cycle Extending on the ideas of Roberts et al. [28], Figure 2.4 show the *goal lifecycle* implemented in CX, defining the program flow based on the states of goals. The progress of a goal in the lifecycle is indicated by a *goal mode*. Upon reasoning, goals are first created in a *formulated* state. Some of the formulated goals will be then *selected* for execution, according to a user defined strategy (encoded by its relation to other goals). Selected goals are *expanded* into plans forming a *simple goal*, or further sub-goals forming a *compound goal*. The goal reasoner selects and *commits* to a plan achieving the goal, and acquires all the ”goal resources” needed for its execution. Then, a goal is *dispatched* for execution by different executives. When the execution of a goal is *finished*, the *goal outcome* indicates whether it had *succeeded* or *failed*. Finally, a goal gets *evaluated* to asses the implication of the outcome (e.g., adapting the worldmodel or creating new goals conditionally).

Goal Interdependence (Goal Trees and Priorities)

In CX, well-defined interdependence relations between goal classes, are used to guide hierarchical or parallel execution of goals. The arrangement of goals in a tree allows to model the different modes of execution as relations between goal classes. Goal *priorities*, guide the goal selection when several goals, with a common parent, are eligible for selection (i.e., the highest priority goal will be selected first). A *compound goal sub-type* defines the relation between goals belonging to a common parent and the semantics of their interaction with their parent (i.e., what the state of executing the sub-goals imply to the state of the parent-goal). A *run-all* sub-type, executes all the sub-goals in parallel, succeeding when all sub-goals succeed (failing if one sub-goal fails). A *try-all* sub-type, executes all sub-goals in sequence, until at one succeeds (failing if all sub-goals fails). A *run-one* sub-type commits to an executable goal (with the highest priority), the outcome of executing the sub-goal directly determines the outcome of the parent. Other compound sub-types exist to control the execution of a single sub-goal. A *retry* sub-type re-tries a sub-goal a number of times. A *timeout* sub-type executes the sub-goal with a time bound.

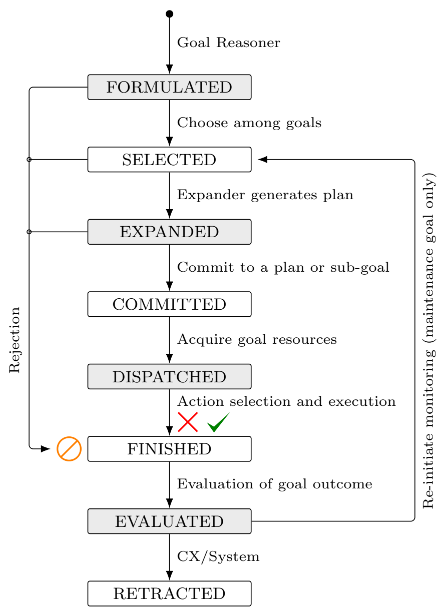


Figure 2.4: Goal lifecycle in CX [25]

The most primitive goal type is a *simple goal* which has no sub-goals, but rather a one or more plans achieving the objective. *Simple goals* only exist at the leaves of the goal tree. A leaf goal is *successful* when all plan actions have been executed successfully. A leaf goal *fails* when an action fails, a temporal constraint is broken or upon intervention by the execution monitoring (e.g., to abort a dispatched goal).

2.6 Incremenetal Goal Reasoning

The goal reasoning model of the current CX agent formulates all RCLL production goals, allowed next by the environment. Each agent selects the best goal it can achieve next, guided by a static goal prioritization. The current approach achieves robustness through the execution of fine-grained goals with short plan lengths, which facilitates recovery from execution failures using a smart execution monitoring strategy. It trades the overhead of scheduling, for fast reaction time and the uncertainty of executing long partial plans (with logical dependencies), for incremental reasoning about the current situation.

The incremental reasoning approach achieves cooperation implicitly, by sharing parts of the agents world model and controlling access to the resources used by their goals with a mutex mechanism. It lacks the deliberation of common high-level goals or an agenda, which utilizes the work done by the agents towards a specific objective. The absence of a common objective makes it hard to decide on the importance of a certain task in varying contexts, resulting in a sub-optimal goal selection strategy. Moreover, the absence of reasoning about expected goal-duration, goal-effect, future goals and availability of resource, results in sub-optimal allocation of tasks to agents and resources, inevitably delaying production. Even worse, low score gaining or even unnecessary work can be overtaken wastefully, while high score gaining opportunities are soon to be executable.

In fact, in order to determine ”if” and ”when” a certain goal should be executed we need to recursively answer questions about its relation to a high-level objective, the causal relation it has with other goals (precedence relations with past and future tasks), when would the goal be executable, how long would it take, which resources are most suitable to achieve it (based on expected availability and locations), which robot is most suitable to execute it (biased on expected availability and travel times between locations).

3 Related Work

First, we review some of the earlier approaches attempted to improve the production in RCLL domain. Then, we review related work relevant to logistics and scheduling in general.

Centralized Task Planner Löbach [17] developed a Centralized Task Planner With Temporal Aspects to employ the ”opportunistic concurrency” arising during single order production in the RCLL domain. Löbach identified the RCLL production as a *simple temporal problem* [4]. They developed a system with architecture similar to CRIKEY [10] , splitting the temporal planning problem into its different start two constituting sub-problems (plan synthesis and scheduling [7]) and solved each separately. Löbach extended the rule-based agent used by the Carologistics team [24] to incorporate temporal aspects by rearranging actions of a sequential plan (synthesized for a single RCLL order) into a Simple Temporal Network (STN). The rearrangement formed a partially ordered plan modeling causal and temporal relations between actions. This allowed for several concurrent execution streams selected greedily by robots.

They presented a PDDL domain and problem models for the RCLL domain ensuring a short planning time. The PDDL problem model is generated online for an order, by dynamically aggregating the world model information. A Fast-Forward planner is then used to find a totally ordered plan for a single actor. Subsequently, an STN generator breaks down the sequential plan (with the aid of an extended PDDL model) into a partially ordered plan representing possible execution streams and preserving temporal and causal relations between actions. The generated STN encodes what actions may be executed concurrently by several actors and the temporal dependency between actions. The STN graph is then used to guide the decision of a greedy agent that picks the most suitable open task from the STN network (i.e., The parentless with the longest execution path).

The system achieves a sub-optimal concurrent behavior, enabling cooperation between robots in producing a single order. The system had the limitation that conflicts in resources allocation between tasks happened, causing a deadlock when executing with multiple agents. Further more, the only concurrency employed is the one found in production tasks of a single order selected by a fixed selection strategy.

OMT Leofante et al. [16] presented an integrated system that uses CX for online execution and monitoring of optimal by construction plans. They encoded the RCLL planning and scheduling problem as a boolean combination of linear constraints over the reals, and employed an OMT (Optimization Modulo Theories) solver to synthesis a high level plan with optimality guarantees. They incorporated domain specific knowledge to gain speed up in solving by explicitly encoding partial orders on actions that have temporal/causal relation in a production pipeline of an single RCLL order. Their approach improved the delivery times of the incremental reasoning agent of the Carologistics team, yet the total robustness of the system was decreased.

Flow Shop Scheduling Perhaps the closest work done to our approach, is that of team Leuphana investigated by Voß et al. [30]. Their approach focused on finding an optimal schedule for a small RCLL instance. They classified the RCLL as flexible flow shop environment and argue that due to the relatively small size of the scheduling problem an optimal approach is promising. They presented a MILP model based on the job shop model of Pan and Chen [27] adjusting it to the requirement of application. To simplify the molding they proposed considering the robots as identical parallel machines and the transportation times as processing time. They then modeled a single robot as a central hub in a blocking flowshop with reentrant processes. $FF7|prec, r_j, d_i, blocking|\sum T_j + E_j$ according to Graham et al. [8], notation. An instance with four C0 jobs and a single robot was solved to optimality in under a minute.

They concluded that the feasibility of scheduling for more complex orders, multiple robots and different optimization criterion still needs to be investigated. They remarked if a model turns to be too complex to solve in real time, an abort criteria has to be defined and other fall back procedures need to be implemented. Their simplified model for single robot does not consider the optimal allocation aspect of robots to tasks. Operations of a job can only be performed by a single robot in a strict sequence. In fact, we argue that formulating the RCLL domain as a *Flexible Flow Shop* is bound to waist concurrency opportunities that could be employed by robots cooperating to produce a single order.

MIPS Scheduling Lütjens et al. [18] developed a MIPS model that solves the Order Batching, Job Shop scheduling and Picker Routing problems in unison for a fleet of picker robots operating in a storage ware house. Their model extends the Capacitance Vehicle Routing Problem with Pick and Delivery by constraints for shared space. Their model has 31 constraints and can solve small sized instances in reasonable time. They developed a three phased heuristic to solve real sized problems. Results showed that their model finds a makespan that is approximately 82% of the greedy task allocation system used by Magazino [1].

[1]https://www.magazino.eu

4 Approach

A centralized goal reasoning and scheduling approach was developed to solve and execute the RCLL production problem; a *scheduling model* integrated within a goal reasoning model. Initially, an *expanded goal-tree* (encoding the production tasks) serves as the primary source for formulating events of the *scheduling model.* A MIP solver, integrated into the *scheduleing model*, solves the scheduling problem. Eventually, the centralized reasoner supervises the execution of the *scheduled goal-tree.*

The centralized goal reasoner splits the production tasks into sub-goals performable by an idle robot (i.e., a *pick* then a *put* sequences intercepted by *machine instructions*, starting and ending with a free gripper). A goal-tree encodes the precedence of production tasks by the child-parent relations between the sub-goals.

The *scheduler* processes an *expanded goal-tree* (the primary input) and resource-meta information (a secondary input), into an *event-based representation.* The *scheduling model* maintains the *event-based representation*, and uses it to communicate with an MIP solver. The event-based representation encodes the scheduling problem as a network of events-nodes, with multiple layers of resource flow. The MIP optimization model is dynamically generated for the network. The optimization model is solved with an objective of minimizing the total make-span of events. The *scheduling model* is post-processed, amending execution-information to the goal-tree. A *scheduled goal-tree* is the result of scheduling an *expanded goal-tree.*

The Job shop scheduling with a *plan selection* and *sequence dependent setup times* is a very hard problem to solve optimally. Yet, for small instances of the problem, optimality can be attained. This work presents a generic MIP formulation which solves a *Flexible Job Shop Scheduling Problem* with *Process Plan Flexibility* (FFJP-PPF) formulated as an event network with multiple commodity flow layers, by simultaneously finding a flow for all resources through the underlying events network, which minimizes the make-span of events. The multilayer network is deduced by parsing a goal-tree and resource-meta information.

We first give an overview of our approach and the structure of the rest the thesis, while introducing a running example and helpful notations used for the rest of this thesis.

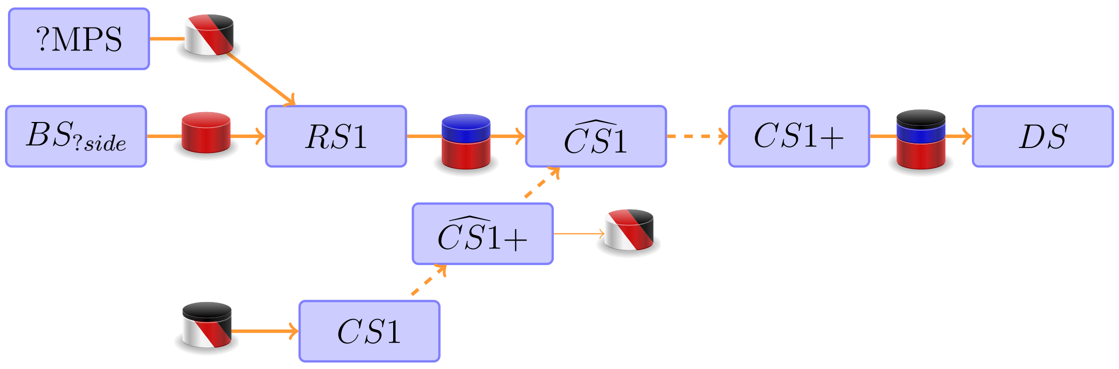


Figure 4.1: C1 order production chains. Edges convey the flow of resources through the production. Curved edges are by-products of operations. Solid edges depict a transportation of a workpiece by a robot, to be processed by the following MPS (square nodes). Dashed edges depict state transformation of a MPS resource.

4.1 Overview

To produce an RCLL order, identical mobile robots has to transport a workpiece between MPS locations. Each MPS is capable of performing a range of production operations. Even though a production workpiece undergoes a strict chain of operations, an MPS might need preparation before performing an operation. Preparing an MPS requires one or more distinct chains of operations.

Figure 4.1 depict the resource flow and the precedence of operations for an RCLL C1 order. The C1 production serves as a running example throughout the reminder of this thesis. An MPS has a fixed location, yet it can be in one of different states. For example, a buffered cap-station ($\widehat{CS}1$) mounts a cap on the production workpiece, only after an empty cap-station ($CS1$) has retrieved a cap ($\widehat{CS}1+$), then cleared of the by-product workpiece (a.k.a., cap-carrier). The precedence of production operations forms an in-tree precedence relations (i.e., each vertex has at most one successor).

Furthermore, several resources might be capable of performing the same operation (e.g., several MPSs could be chosen to produce the material used to fill a ring station; several identical robots can perform a transportation operation). A resource might require a setup period between two operations, dependant on the sequence of operations. Our motive is to minimize the production time, by determining, when would, each resource (i.e., MPS or robot) perform which operation.

Our scheduler attempts to minimize the total production duration that is needed to execute a goal-tree. It encodes the goal-tree as a MIP optimization problem, upon which solving yields a goal-execution schedule and resource allocation schedules (i.e., a schedule for each resource). Our centralized goal reasoning and scheduling approach has the following main stages:

1. Initially, resources are declared (PDDL resource objects; resource meta-information).
2. The reasoner *formulates a goal-tree*, modelling the production tasks
3. The reasoner *expands each sub-goal*, using a plan library. A plan is formulated for each available resource allocation
4. The scheduler *pre-processes* the plans to deduce *resource usage-requirments*
5. The scheduler constructs the *event-based representation*, encoding the scheduling problem as a network of events, with multiple layers of resource flow
6. The MIP solver is called to generate and solve the optimization problem
7. The scheduler *post-processes the solution* and amends the goal-tree with execution-information
8. The reasoner *executes the scheduled goal-tree*; sub-goals are dispatched according to an execution and resource allocation schedules
9. The reasoner dispatches and monitors individual actions, to be remotely executed by (an allocated) distributed agent

This section describes the main contribution of this thesis.

Section 4.2 describes the declaration of resources and their resource meta-information. Section 4.3 describes the formulation and expansion of a *goal-tree* (the input of the scheduler), and the execution of a *scheduled goal-tree* (the output of the scheduler). Section 4.4 explains how the scheduler parses the goal-tree into an *event-based representation*, and formulates a corresponding MIP model. Section 4.5 covers the implementation aspects and extensions to the goal reasoning model and its *goal life-cycle*; implementation aspect of the scheduling model and its interaction with the reasoner and the solver, during its *schedule life-cycle*. Finally, execution of the goal-tree by distributed agents is described in more depth.

4.2 Resource Declaration

A PDDL super-type `resource` is dedicated to describe resource objects. A resource object can only be used by a single running plan at any given time. In the RCLL domain, PDDL types `robot`, `mps` and `workpiece`, belong to the `resource` super-type. Listing 4.1 shows the resources declared in PDDL for the RCLL domain. An instance of a resource (e.g., a cap-station or Robot-1) is used *exclusively* by (at most) a single running plan (e.g., `MOUNT-CAP`, `FILL-CAP`) at any given time.

```
(:types
    resource - object
    robot - resource
    mps - resource
    workpiece - resource)
```

Listing 4.1: Resource objects deceleration in the RCLL PDDL domain . A resource is an object which should only be used by a single running

```
 (resource-info (type robot)
                (consumable FALSE)
                (producible FALSE)
                (setup-preds (create$ at))
                (state-preds (create$ can-hold is-holding)))
```

Listing 4.2: Resource-meta information of a `robot` PDDL object, declared in CLIPS. `setup-preds` and `state-preds` specify the PDDL predicates names used to describe a resource-state and resource-setup. The `consumable` and `producible` fields specify whether the resource is available by the environment, or whether it is consumed (or produced) as a result of executing an action.

At plan start, a resource is used (i.e., consumed) at an expected *state* and *setup*; at plan end, a resource is released at known *state* and *setup*.

Resource state is a set of PDDL predicates which are used to describe and distinguish different possible *states* of a resource instance (such as, the gripping status of a robot; the colors mounted on a workpiece; the buffering status of a cap-station). A plan consumes and (or) produces a resource at specific a state.

Resource setup is a set of PDDL predicates which are used to describe a property of a resource instance, which is trivial to change (such as, a robot location). A plan consumes and (or) produces a resource at specific a setup.

Some resources are *produced* or *consumed* as a result of executing a plan action (such as dispensing or delivering a workpiece), while others are available as a commodity provided by the environment (such as, robots and MPSs). The information specifying whether or not a resource type is *producible* (or *consumable*) by plan actions, as well as, which PDDL predicates are used to describe *states* and *setups* of a resource, are specified as *resource-meta information*. For example, resource-meta info. for a `robot` is specified in Listing 4.2

4.3 The Goal Reasoner

Our *scheduling model* is integrated within a centralized goal reasoning model. An expanded goal-tree (encoding the scheduling problem) serves as the primary source of information for generating the *scheduling model*. After scheduling, the centralized reasoner supervises the execution of the *scheduled goal-tree*.

Initially, a goal-tree is formulated, encoding the production tasks and their logical precedence. A sub-goal is expanded using a plan library, by formulating a plan for each possible resource allocation. An expanded goal-tree (encoding a scheduling problem) serves as the primary source of information for the *scheduler*. The *scheduler* formulates and uses a *scheduling model* to solve the scheduling problem. Finally, the *goal-tree* is amended with execution information, to become a *scheduled goal-tree* (i.e., plan selections, sub-goal start-times, and setup sub-goals formulation).

Eventually, the *scheduled goal-tree* is executed in compliance with sub-goal start-times and resource allocation-schedules (i.e., an allocation-schedule for each resource). A sub-goal is achieved by executing a selected plan which is started at a specific time. Any unexpected temporal violations during execution are remedied by complying with the resource allocation-schedules. This protects the execution from any potential logical threats, which result from violating the temporal schedule.

4.3.1 The Expanded Goal-Tree

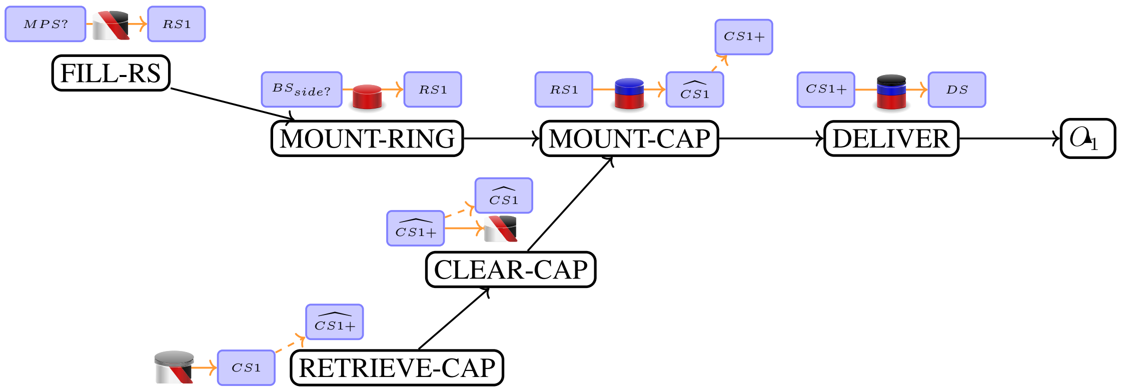
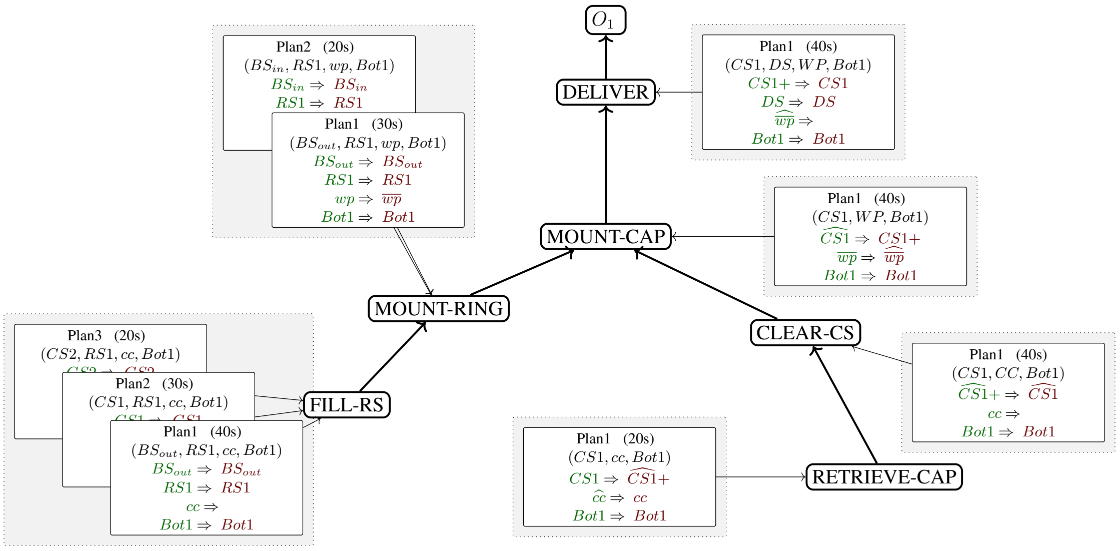


Figure 4.2: A centralized goal-tree formulated for a C1 order. It breaks down the C1 production chain into sub-goals which require an allocated agent (free to grip). A sub-goals models a *pick*, *move*, *put* sequence, intercepted by communicating with the machine.

Initially, a goal-tree is formulated encoding the production tasks. Tasks are broken down into sub-goals that require the allocation of an idle robot with a free gripper. A sub-goal consists of a *pick*, *put* sequence, intercepted by sending *machine instructions*. Figure 4.2, shows a goal-tree modeling the C1 order of Figure 4.1. It can be seen that a sub-goal roughly corresponds to a group of production operations in Figure 4.1. A goal-tree encodes the precedence of the production tasks by the child-parent relations between the sub-goals.



O_1
DELIVER
MOUNT-CAP
MOUNT-RING
FILL-RS
CLEAR-CS
RETRIEVE-CAP
Plan2 (20s) $(BS_{in}, RS1, wp, Bot1)$ $BS_{in} \Rightarrow BS_{in}$ $RS1 \Rightarrow RS1$
Plan1 (30s) $(BS_{out}, RS1, wp, Bot1)$ $BS_{out} \Rightarrow BS_{out}$ $RS1 \Rightarrow RS1$ $wp \Rightarrow \overline{wp}$ $Bot1 \Rightarrow Bot1$
Plan1 (40s) $(CS1, DS, WP, Bot1)$ $CS1+ \Rightarrow CS1$ $DS \Rightarrow DS$ $\widehat{\overline{wp}} \Rightarrow$ $Bot1 \Rightarrow Bot1$
Plan1 (40s) $(CS1, WP, Bot1)$ $\widehat{CS1} \Rightarrow CS1+$ $\overline{wp} \Rightarrow \widehat{\overline{wp}}$ $Bot1 \Rightarrow Bot1$
Plan3 (20s) $(CS2, RS1, cc, Bot1)$
Plan2 (30s) $(CS1, RS1, cc, Bot1)$
Plan1 (40s) $(BS_{out}, RS1, cc, Bot1)$ $BS_{out} \Rightarrow BS_{out}$ $RS1 \Rightarrow RS1$ $cc \Rightarrow$ $Bot1 \Rightarrow Bot1$
Plan1 (40s) $(CS1, CC, Bot1)$ $\widehat{CS1}+ \Rightarrow \widehat{CS1}$ $cc \Rightarrow$ $Bot1 \Rightarrow Bot1$
Plan1 (20s) $(CS1, cc, Bot1)$ $CS1 \Rightarrow \widehat{CS1}+$ $\widehat{cc} \Rightarrow cc$ $Bot1 \Rightarrow Bot1$

Figure 4.3: Expanded Goal-tree for C1 order. A grounded plan is generated for each possible resource allocation.

A sub-goal is expanded using a plan library, by formulating a plan for each available resource allocation. Figure 4.3 shows an *expanded goal-tree* which is generated for a *single* available robot. For example, a `FILL-RS` goal has a plan formulated for each possible allocation of a *robot* and each possible *picking* MPS-location (i.e., base-station, cap-station-1 shelf and cap-station-2 shelf). Plans of a goal have varying durations as a result of using alternative resource sets or action sequences. A plan consumes a resource at a known starting state; *exclusively* uses the resource for the plan duration, then releases it an an end state. The resource allocation decision is elevated to become a plan selection decision (i.e., the selection of a plan determines the resources allocated to a goal).

4.3.2 The Scheduled Goal-Tree

A *scheduled goal-tree* is the result of scheduling an *expanded goal-tree*. After the scheduler determines execution and allocation schedules (i.e., an execution schedule for sub-goals, and an allocation schedule for each resource), the expanded goal-tree is prepared for execution, by amending the sub-goals with the scheduled execution-information. For each sub-goal:

- A plan is selected
- Execution times are set

- Sub-goals are formulated, if a resource setup is needed

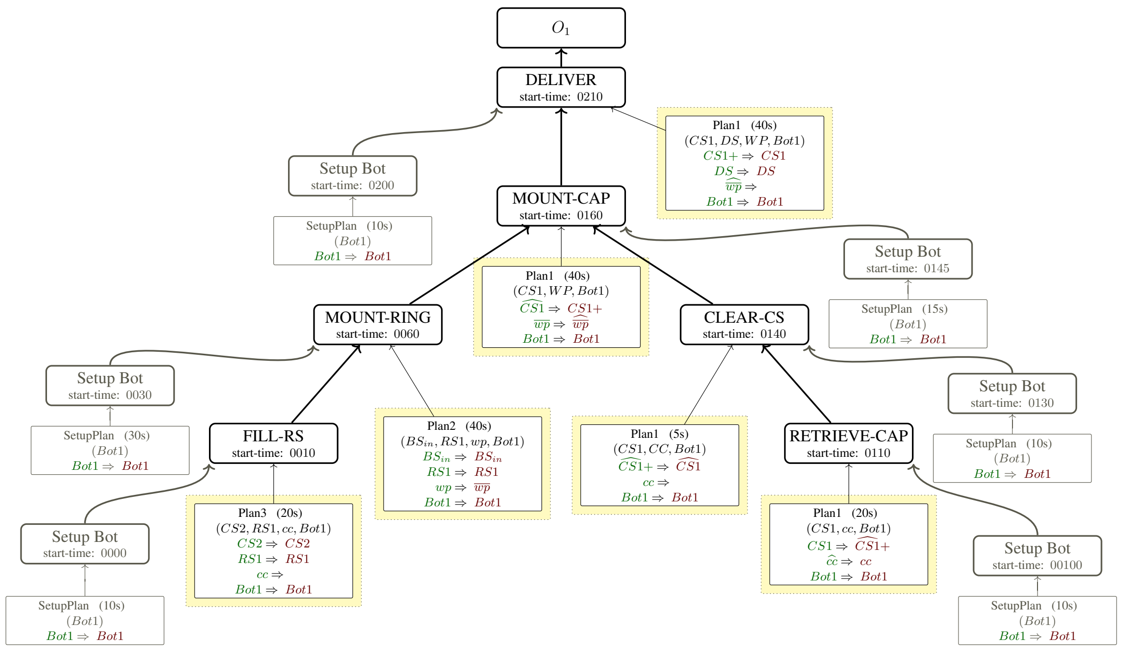


Figure 4.4: A *scheduled goal-tree*; the result of scheduling an *expanded goal-tree*. The tree is amended with execution-information and is ready for execution. A plan has been selected (highlighted), setup-goals were formulated (brownish sub-goals), and a start-time was set for each sub-goal.

Figure 4.4 shows the result of amending the execution-information to the *expanded goal-tree* in Figure 4.3 (the C1 running-example). Setup goals for robots are created as sub-goals, to prepare the robot for the upcoming goal. The is *scheduled goal-tree* is ready for execution.

4.3.3 Execution

A goal is achieved by executing a selected plan. The plan is started when:

- The goal's start-time is reached
- Its sub-goals are completed
- For each of the resources required by the goal, the goal is the next scheduled allocation appearing in the resource allocation-schedule.

The central reasoner dispatches individual actions for remote execution by their (allocated) distributed agents. The central reasoner monitors the execution for any temporal or logical violations (i.e., schedule or plan violations) resulting from execution uncertainty. If a temporal violation occurred (e.g., a sub-goal took longer than expected), the above execution scheme ensures that the execution will continue with a delay (without breaking any logical precondition). Indeed, complying with the resource allocation-schedules as well as the precedence of sub-goals during execution, protects the execution from any potential logical threats, that results from executing actions earlier or later than expected.

4.4 The Scheduler

The *scheduling model* consists of an *event-based representation* of the scheduling problem, used to communicate with an MIP solver. The *event-based representation* is generated by parsing an *expanded goal-tree* (the primary input) and resource-meta information (the secondary input). The *scheduler* uses a MIP solver to solve the scheduling problem. Eventually, the *scheduler* amends the goal-tree with execution-information. A *scheduled goal-tree* is the result of scheduling an *expanded goal-tree.*

First, the goal-tree and the resource-meta information are **pre-processed** in order to deduce *resource-usage requirements.* The resource-usage requirements specify the consumption and production requirements of the resources, by each plan. The scheduler then **processes** the goal-tree as well as the resource-usage requirements, to formulate an *event-based representation* of the scheduling problem. The event-based representation encodes a scheduling problem (in a *scheduling model*), analogical to how a goal encodes a planning problem (in a goal reasoning model). The *event-based representation* encodes the sub-goals and plans as events in a multilayer commodity flow-network.

Analogical to goal expansion into plans by calling a planner, the event-based representation is expanded into scheduled events by calling an MIP solver. An optimization problem is generated based on the event-based representation. Simultaneously finding the flow for all the resources between the events of the *multi commodity flow temporal network*, which minimizes the total make-span of events, yields start-times for events and an allocation schedule for each resource. Finally, the solution is post-processed by the scheduler, to update the *event-based representation* and eventually amend the goal-tree with execution information.

4.4.1 Pre-processing

The expanded goal-tree (primary scheduler-input) and the resource-meta information (secondary scheduler-input) are pre-processed to deduce resource-usage requirements. The resource-usage requirements specifies the consumption and production requirements of each resource by each plan.

At a plan's start, a resource is used at a specific resource-state and resource-setup (i.e.,a consumption requirement). At a plan's end, a resource is released at a specific state and setup (i.e., a production requirement). The PDDL predicates which are used to describe the resource-states (and setups), are specified as a resource-meta information, during resource declaration. The resource-usage requirements (i.e., consumption and production requirements) are deduced from plans preconditions and effects before formulating the *event-base representation.*

Plan preconditions is the set of preconditions of plan-actions, which are not satisfied by an effect of an earlier action in the plan.

Plan effects is the set of effects of plan-actions, which are not negated by an effect of a later action in the plan.

Consumption requirement of a resource by a plan is the set of resource-state and resource-setup propositions, appearing as a precondition of the plan.

Production requirement of a resource by a plan is the set of resource-state and resource-setup propositions, appearing as an effect of the plan.

A plan consumes a resource at a known consumption requirement by *exclusively* using the resource for the plan duration, then produces the resource at a known production requirement. After being released by a plan, some resources need a *setup duration* before being available for use by the following plan. The setup duration depends on the sequence by which the resource is allocated to two consecutive plans. For example, a robot allocated to a `CLEAR-CAP` goal followed by a `MOUNT-RING` goal needs a different setup duration than if it was followed by `MOUNT-CAP` (i.e., traveling from the cap-station output to the base-station, as opposed to traveling from one cap-station side to the other). Resource setup duration between all possible plan sequences are calculated apriori.

4.4.2 Event-based representation

The event-based representation encodes a scheduling problem (in a *scheduling model*), analogical to how a goal encodes a planning problem (in a goal reasoning model); a goal is expanded into plans (potentially) by calling a planner, similarly an event-based representation is expanded into scheduled events by calling an MIP solver.

The scheduler parses an expanded goal-tree (the primary input) ,as well as, its resource-usage requirements and setup-durations (the pre-processed information) in order to formulate an event-based representation of the scheduling problem. The event-based representation is a multilayer network of events. It consists of an event-precedence base graph, and multiple layers of commodity flow (i.e., a layer for each resource)

Precedence graph is a directed acyclic graph, where goals and plans (of an expended goal-tree) are parsed into event-nodes connected by directed edges of event-precedence relations. Figure 4.5, shows a small part of the precedence graph which is generated by parsing the expanded goal tree in Figure 4.3 (i.e., the expanded goal-tree of the C1 running-example). A precedence graph serves as the base-graph for a multilayer network; each layer represents the commodity flow of a resource.

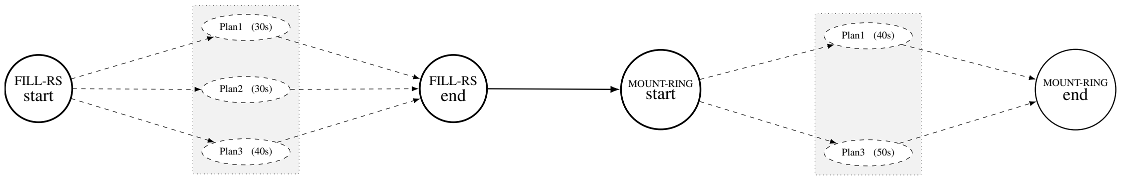


Figure 4.5: Goal events and plan events in the precedence base-graph, parsed from the goal-tree. A start and an end events are created for each sub-goal (solid nodes). Parent-child relation between the sub-goal are depicted with the directed (solid) precedence edges. An event is created for each plan (dashed elliptical nodes). Only a single event in a selection group (depicted by the gray scopes) shall be included in the scheduled solution.

Commodity flow-network layer is a directed cyclic flow-network, where directed weighted edges represent the flow of a resource between events of the base-graph (i.e., the precedence graph). The resource-usage requirements are used to determine which events should be connected by a flow edge. The setup durations are used to determine the weights of the a flow edges.

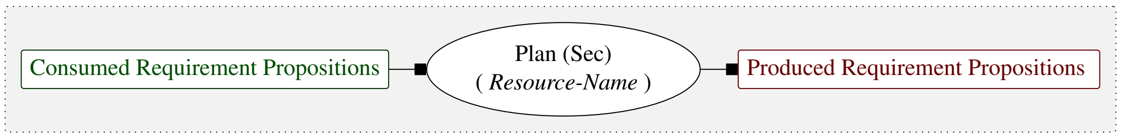

Consumed Requirement Propositions
Plan (Sec)
(Resource-Name)
Produced Requirement Propositions

Figure 4.6

Figure 4.6 shows how the usage requirements of a resource are attached to an event node (e.g., a plan-event), in a commodity flow layer. The usage-requirement labels (square boxes) serves as a proxy for connecting the resource flow edges to the event-node (in a commodity flow layer). The green label (consumption requirement propositions) proxies the in-flowing edges of a resource-type; the red label (production requirement propositions) proxies the out-flowing edges.

Figure 4.7 shows the resource-usage requirements of a `MOUNT-CAP` plan (4.7a), parsed into an event-node with multiple requirement labels (in the multilayer network). Each pair of green and red labels of a resource-requirement (i.e., consumption and production requirements) appears in a resource flow layer of the multilayer network.

After being consumed during an event, a resource is produced (by the event) in a known state. The state of a resource (before and after an event) is encoded as a set of state

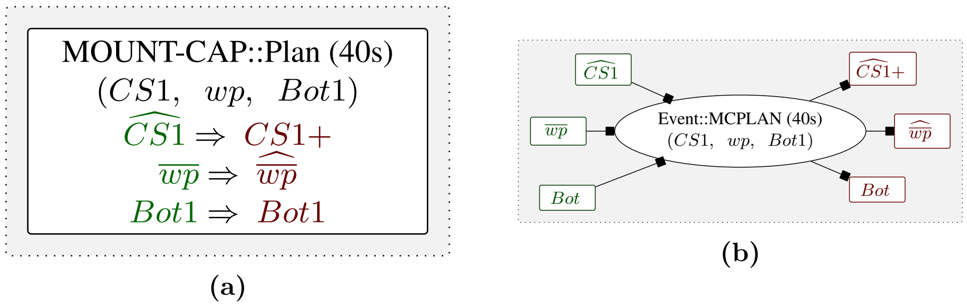


Figure 4.7: A mount-cap plan 4.7a parsed into a base-graph event with requirement labels in the multilayer network 4.7b. The state symbols (defining the requirements) are shorthand for known resource-state propositions.

propositions attached to the flow-edges (i.e., an *edge-state*). Figure 4.8 shows flow edges (with different *edge-states*) connected to an event-node. Only edges that satisfy the consumption requirement of an event are allowed to flow into the event (i.e., the *edge-state* propositions satisfies the resource consumption requirement propositions; analogical to PDDL actions preconditions). The state of an out-flow edge (i.e., a resource produced by the event) is the accumulative state of a corresponding in-flow edge, altered by the production requirements of the event (i.e., analogical to applying effects of a PDDL action; the state propositions of an out-flow edge are the propositions of an in-flow edge after the resource consumption requirement has been applied).

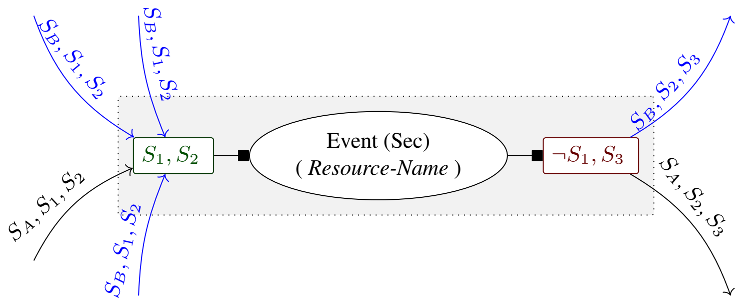


Figure 4.8: The flow of a resource with different states propositions (i.e., *edge state*) into and out of an event. The in-flow is connected to the event through the consumption proxy (green label); inflowing edges satisfy the consumption-requirement propositions (S_1 and S_2). The out-flow is connected to the event through the production proxy (red label); outflowing edges satisfy the production-requirement propositions ($\neg S_1$ and S_3). State propositions, which are not changed by the event are preserved before and after the event, ensuring the continuity of resource states (e.g., S_B for the blue edges, and S_A for the black edges). An inflowing edge and an outflowing edge with the same color are selected in the solution.

The edges and nodes labeling scheme ensures the continuity of a resource state, before and after an event (e.g., the black in-flow edge in Figure 4.8, only causes the black out-flow edge, while the blue in-flow edges only causes the blue out-flow edge). Figure 4.9

shows an example of a cap-station resource, used by a `FILL-RS` plan; the plan fills a ring-station with a cap-carrier, picked from the shelf cap-station. The plan-event does not have any usage requirements for the cap-station (hence, the empty node labels), since the plan uses the cap-station without changing its state. Nevertheless, a resource state has to be preserved before and after being used by the event (resource state continuity). The black and blue edges depicts the possible flows of the different resource-states of a cap-station, before and after it is consumed by the `FILL-RS` event. (i.e., If a black in-flowing edge is selected in the solution, the black out-flow edge has to also be selected (i.e., continuity of resource-states).

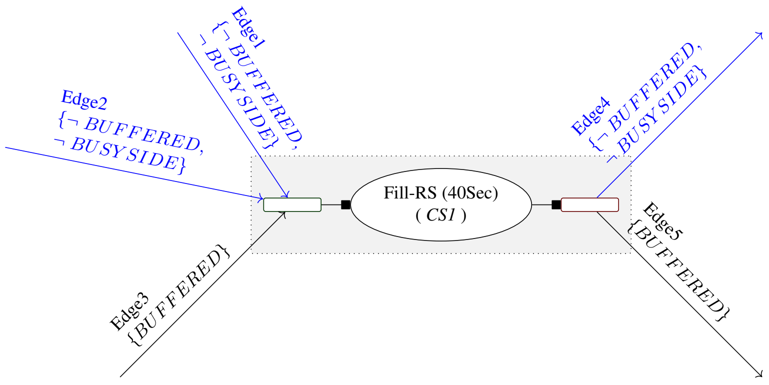


Figure 4.9: The flow of cap-station, with different possible states, into a fill-rs event. The event has no usage-requirements (hence, empty requirement labels). If edges 3 is selected in the solution, edge 5 will also be selected. If either of edges 1,2 is selected, edge 4 will be selected.

When a setup duration is required between two events, weights of durations are attached to the flow edges. Figure 4.10 shows part of the flow nextork layer of a cap-station, parsed from the events producing a C1 order (parsed from the expanded goal-tree in Figure 4.3); all events which uses the cap-station are shown, except for the delivery event. Since the cap-station is a resource available by the environment (i.e., declared as such in the resource-meta info, as opposed to being producible or consumable as a result of executing a plan action), a *source* and a *sink* event are created to depicted the initial production and the final consumption of the resource by the system. Source and sink events have usage-requirements similar to plan-events. The colors of the flow edges depicts different *edge-states*. Only a single inflowing and a corresponding outflow edge to an event shall be selected in the solution-schedule.

4.4.3 MIP Model

The *event-based representation* encodes a scheduling problem as a commodity flow multilayer network of event nodes (i.e., an event-precedence base-graph and resource flow layers). The *scheduling model* uses an MIP solver to expand a solution-schedule (i.e., schedule the events). The *scheduler* uses the event-base representation to formulate the

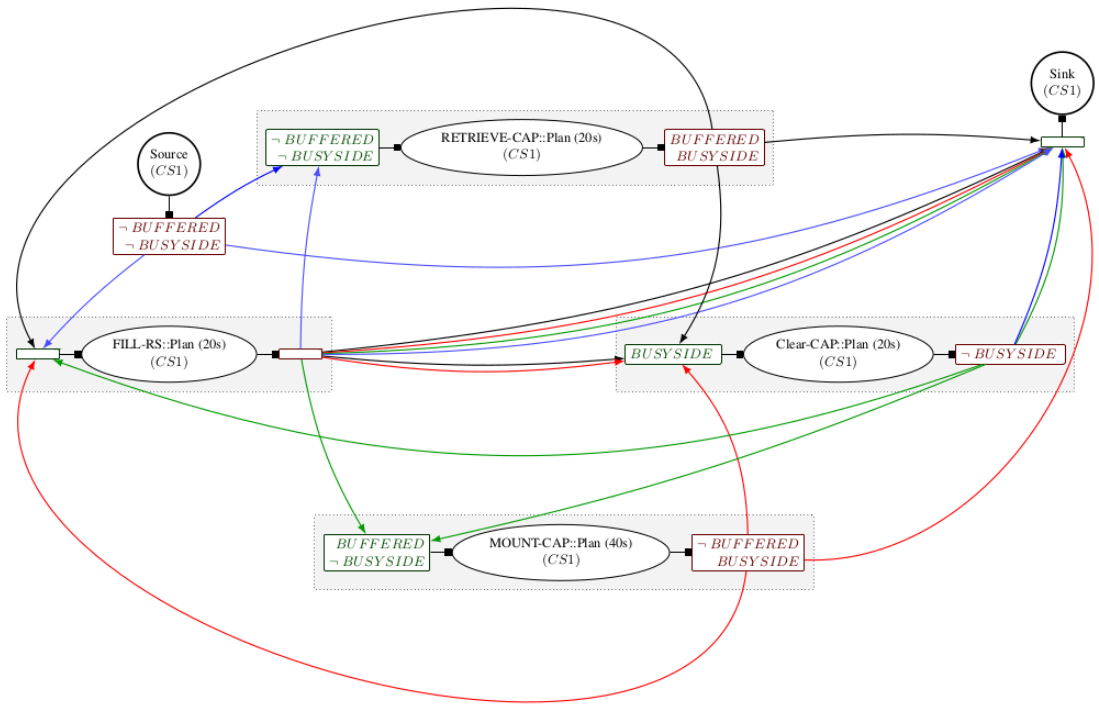


Figure 4.10: Part of the flow network layer of a cap-station resource, between the events of the base precedence-graph. The events are parsed from a C1 expanded goal-tree. The edge colors depicts the different states of the resource (i.e., *(*edge-state)). An edge flow into an event in a state (satisfies the event preconds), and release at a (potentially) different state (i.e., the event applies a conversion function to the edge-state; analogical to PDDL preconds and effects). Only a single edge is allowed to flow-in or out of an event. That state of the in-flowing edge determines the state of the out-flowing edge. The state of an edge can be read from the event (labels) it connects.

MIP optimization problem and triggers the solver to find an optimal (or near optimal) solution.

First, the scheduler uses the event-based representation to populate the data sets used to build the MIP model. MIP model generation and optimization are consequently trigged. The objective to minimize the makespan of events. The result of the optimization is a *start-time* for each event, wether an event is *selected* in the schedule and the *sequence* by a resource is allocated to events.

The MIP model relies on three atomic entities, directly deduced from the event-based representation. Namely *events*, *selectors* and *resources*. Each entity can be linked to one of the three main decisions, which our scheduler aims to determine. The *times* of events, the *selection* of events and the *sequence of events* served by each resource.

Event-times T_e

The decision variables T_e indicates the *start-time* of an event e. Where e is in the set of atomic events E. Event duration δ_e and precedence relations $\alpha_{e1,e2}$ are known apriori.

$E = \{1, ..n\}$	Set of atomic events
δ_i	Duration of event $i \in E$
$\alpha_{i,j} \in \{0, 1\}$	Wether event i precedes event j, $(i, j \in E)$

Selectors Z_s

A selection variable is used to groups sets of events or edges into selection groups, and gain control over their selection in the final solution. The decision variable Z_s indicates the *selection* of a selection group s. A selector s has an initial selection state ω_s, known apriori.

$S = \{1, ..n\}$	Set of all selection groups (i.e., selector)
$\omega_s \in \{0, 1\}$	Initial selection status of a selector $s \in S$
$SE_s \subset E$	Subset of events selectedable by selector $s \in S$

An initially selected selector will always be included in any valid solution

$$\omega_s(1 - Z_s) \leq 0 \qquad \forall s \in S \tag{4.1}$$

A selector could additionally be used to group other selectors, forming a more complex (select-one or select-all) selection relations (i.e., a selection group of selectors).

$SG_s \subset S$	Subset of selectors $\subset S$ forming a run-one selection group; $s \in S$, $s \notin SG_s$

Exactly one selector is selected from a *select-one-of* group , iff the group selector s is selected. This models a *select-one-of* relation between the selectors in SG_s. For example, this is used to model that only a single plan-event can be *selected* for a *selected* goal-event; in Figure 4.5, only a single plan-event (dashed elliptical nodes) is selected in a *select-on-of* selection group (gray scopes)

$$\sum_{si \in SG_s} Z_{si} = Z_s \qquad \forall s \in S \tag{4.2}$$

Whenever there is a precedence relation between two events $i \prec j$ (i.e., $\alpha_{i,j} = 1$), the later event j should start after the duration of the earlier event i. This should only be the case, iff both of the events are selected (i.e., $Z_{si} = 1, Z_{sj} = 1$). An event i is considered selected, iff a selector Z_{si} of a the group it belongs to $i \in SE_{si}$ is selected. Constraint (3) is used to model *goal* $\prec$ *sub-goal* and *goal* $\prec$ *plan* precedence relations between events. For example, the precedence edges (dashed and solid) of the precedence graph in Figure 4.5. Constraint (3) uses the big M notation to model an implication (indicator) relation.

$$\alpha_{i,j}\left(-T_j + T_i + \delta_i\right) \leq M(2 - Z_{si} - Z_{sj}) \quad \forall i, j \in E \; \{i \in SE_{si},\; j \in SE_{sj}, si, sj \in S\} \tag{4.3}$$

Resource allocation sequence $X_{r,i,j}$

The decision binary variable $X_{r,i,j}$ models the decision that a resource r is used (i.e., consumed) by event j, directly after being released (i.e., produced) by event i. The event i belongs to the sub-set of events which produces r, EP_r. The event j belongs to the sub-set of events which consumes r, EC_r. An events uses and/or releases a resource r in a quantity $\Theta_{r,e}$ which is known aprior.

$R = \{1, ..n\}$	Set of resources
$EC_r \subset E$	Subset of events which consumes a resource $r \in R$
$EP_r \subset E$	Subset of events which produce a resource $r \in R$
$\Theta_{r,i}$	Quantity required of a resource $r \in R$ by event $i \in E$

After being released by an event $i \in EP_r$, a resource requires a *setup duration* before it becomes available for the next event $j \in EC_r$. The *setup duration* $\Delta_{r,i.j}$ depends on the sequence at which the events i and j are chosen. Setup durations for all possible *(producer-event* $\prec$ *consumer-event)* sequences of r are calculated apriori.

$\Delta_{r,i,j}$	Setup duration needed by resource $r \in R$ to serve event $i \in E_r^C$ directly after serving event $j \in E_r^C$

Whenever a sequence of events ($i \prec j$) is allocated to r (i.e., $X_{r,i,j} = 1$), the later event j can is scheduled to take place after the earlier event i had finished, in addition to any *setup duration* needed by r (to become available, between the events i and j).

$$-T_j + T_i + \delta_i + \Delta_{r,i,j} \leq M(1 - X_{rij}) \quad \forall r \in R,\; \forall i \in EP_r,\; \forall j \in EC_r \tag{4.4}$$

We assume an event $j \in EC_r$ can consume only a single unit of resource r through any chosen sequence $X_{r,i,j}$ (i.e., $X_{r,i,j}$ has a capacity of 1 unit of r). Whenever an event $j \in EC_r \cap ES_s$ is selected (i.e., $Z_s = 1$), the event has to reach its consumption capacity $\Theta_{r,j}$ of resource r. This means that the number of sequences $X_{r,e,j}$ chosen in the solution, where the event j appears as a consumer, has to equal the consumption capacity of the event. (i.e., the total in-flow of r to a *selected* event j, has to be equal to its consumption-capacity

$$\sum_{i \in EP_r} X_{rij} = Z_s \,.\Theta_{rj} \qquad \forall j \in EC_r \cap SE_s,\ r \in R,\ s \in S\} \tag{4.5}$$

We assume an event $i \in EP_r$ can produce only a single unit of resource r through any chosen sequence $X_{r,i,j}$ (i.e., $X_{r,i,j}$ has a capacity of 1 unit of r). Whenever an event $i \in EP_r \cap ES_s$ is selected (i.e., $Z_s = 1$), the event has to reach its production capacity $\Theta_{r,j}$ of resource r. This means that the number of sequences $X_{r,e,j}$ chosen in the solution, where the event i appears as a producer, has to equal the production capacity of the event. (i.e., the total out-flow of r from a *selected* event i, has to equal to its *production-capacity*

$$\sum_{j \in EC_r} X_{rij} = Z_s \,.\Theta_{ri} \qquad \forall\ i \in EP_r \cap SE_s,\ r \in R,\ s \in S\} \tag{4.6}$$

Finally, the completion time of an event $e \in E$ should be less than the completion time of the schedule

$$T_i \leq T_{Max} \qquad \forall i \in E \tag{4.7}$$

This resource-allocation modelling does not account for different resource-states. It merely assumes that each resource producer is connected by a single edge to the each resource consumer. This model has been extended to represent different edges for different resource-states. Events are only connected by an edges occasionally (iff the *edge state* satisfies the usage-requirements of the events it connects). The event-based model is responsible for generating the edges, the MIP model was extended to encode multiple potential edges between the events, and *flow continuity* of edge-states inflowing and out-flowing to an event.

4.5 Implementation

This section describes the implementation aspects of our approach. It explains the extension to the life-cycles of the goal reasoning model; the life-cycle of the newly developed scheduling model and how the event-based model is formulated and used to populate the MIP formulation.

4.5.1 Goal Reasoning Model

Our approach employs a goal reasoning model to serve two main functions. Firstly, it provides a vessel for encoding the tasks needed to achieve a desired objective in the PDDL domain. Tasks and their logical relations are formulated into a goal-tree. Secondly, the goal reasoning model is used to execute the goals, according to a time schedule and resource allocation schedules, without violating any precedence relations encoded by the goal-tree.

After formulation and expansion of the goal-tree, the information coded by the goal-tree is parsed to generate a *scheduling model*, which is used to communicate with the MIP solver. Finally, the *goal-tree* is amended to reflect the solution-schedule, and execution is started. This behaviour is by implemented by extending to the goal-reasoning model of [25].

Model Extensions

Our task-level encoding and scheduled execution relies on the following extensions made to the goal reasoning model of the Clips-Executive.

- A goal can commit to plans and sub-goals simultaneously.
- A new goal sub-type (i.e., `SCHEDULE-SUBGOALS`) was developed, to encode a precedence-relation between two goals as well as execute the sub-goals in compliance with a sub-goal start-times, the precedence of sub-goals and resource allocation schedules.

Goal Life-cycle

A goal *model* tracks the life-cycle of a goal throughout its different stages. Furthermore, a goal sub-type defines the interaction between a goal and its children (i.e., sub-goals and plans) during the life-cycle. For `SCHEDULE-SUBGOALS` sub-type the following behaviour is implemented. A goal is:

- **Formulated** when its parent has been selected
- **Selected** when all of its sub-goals are selected (i.e., top-down selection)
- **Expanded** when all of its plans and selected sub-goals were expanded (i.e., bottom-up expansion)

- **Committed** when all its sub-goals have been *completed* (i.e., bottom-up commitment); the goal start-time has been reached; no other goal requires any of it's required-resources, earlier than this goal. Upon committing, a goal attempts to acquire the required-resources
- **Dispatched** when all of its required-resources have been acquired. The goal is then consequently executed by starting a plan

A goal remains in an *expanded* state till its *start-time* is reached. Start-times are determined as a result of scheduling or any other temporal reasoning process.

4.5.2 Scheduling Model

As the scheduler is integrated within a goal reasoner, the *scheduling model* can be viewed as sub-model, within the *goal-reasoning model.* The *scheduling model* acts as an intermediate layer between *goal-reasoning* and scheduling (for example, using a MIP solver), by encoding the scheduling problem and interacting with the solver.

Similar to a goal life-cycle, the *scheduling model* relies on a *life-cycle* of a `schedule` instance, keeping the scheduling process independent from the goal reasoning; the entire *life-cylce* of a schedule instance takes place during goal expansion. This decoupling has two main advantages. First, it allows several scheduling processes with different scheduling parameters to be started simultaneously, for the same goal-tree. Furthermore, it helps maintain transparency to the technology used by the scheduler. Indeed, the MIP solver could be easily altered or replaced by another scheduler without influencing the interaction between the goal reasoning and the scheduling models.

Event-based representation

The *scheduling model* parses a fully expanded *goal-tree* to a corresponding homogeneous *event-based representation.* Entities and relations encoded previously by the tree (e.g., goals, plans, resources and resource-requirements) are parsed into a flatter representation of `events` and a `edges` between the events.

Schedule is the entity central to *scheduling model*, modelling an instance of a scheduling process (analogical to a goal in the goal reasoning model). The schedule mode tracks the *life-cycle* of the schedule through the scheduling process. Listing 4.3 show the definition of a schedule.

```
(deftemplate schedule
     (slot       id            (type SYMBOL))
     (slot       mode          (type SYMBOL))
     (slot       solver-status (type SYMBOL))
     (multislot  solver-params (type SYMBOL))
```

```
(deftemplate schedule-event
    (slot  sched-id   (type SYMBOL))
    (slot  id         (type SYMBOL))
    (slot  entity-id  (type SYMBOL))
    (slot  entity-at  (type SYMBOL)  (allowed-values START END))
    (slot  duration   (type INTEGER) (default 0))
    (slot  lbound     (type FLOAT)   (default 0.0))
    (slot  ubound     (type FLOAT)   (default 1500.0))
    (slot  scheduled  (type SYMBOL)  (allowed-values TRUE FALSE))
    (slot  scheduled-start (type INTEGER) (default 0))
)
```

Listing 4.4: Definition of an atomic event, in the event-based representation of the *scheduling model*. An event tracks of an identifier for the entity it belongs to (such as, a plan, a goal or a resource); whether the event is selected (i.e., scheduled); start-time and duration of the event. Upon *formulation*, the duration and the *initial* selection status are specified. Upon *expansion* (by calling the solver), the *start-time* and wheter the event is *eventually* scheduled in the solution, are determined.

```
    (multislot  goals         (type SYMBOL))
    (multislot  resources     (type SYMBOL))
    (multislot  dispatch-time (type INTEGER)))
```

Listing 4.3: Definition of a schedule, the entity central to the *scheduling model*. A schedule tracks the *life-cycle* of a schedule instance through the scheduling process, via its mode; which goals and resources the scheduling is performed for; any solver parameters used to influence the solver's algorithm or define solving strategies (e.g., define time-limit or minimum solution gap). The schedule id is referenced by elements of the events-based representation , belonging to the same schedule instance.

Event is an atomic event in a schedule instance. (e.g., a goal start, goal end, resource source, resource sink or plan events). Listing 4.4 shows, the definition of a schedule event.

Precedence is a relation between events $a \prec b$, modelling that *" event a finishes before event b starts"*. For example, a precedence relation exists between a goal and each of its children (i.e., goal $\prec$ sub-goals and goal $\prec$ plans).

Usage requirement is a relation that exists between an event and a resource. An event consumes (or produces) a resource at a specific *resource-state* and *resource-setup*. Listing 4.6, shows the definition of a schedule's usage-requirement

```
(deftemplate usage-requirement
   (slot sched-id (type SYMBOL))
   (slot event-id (type SYMBOL))
   (slot resource-id (type SYMBOL))
   (slot usage-units (type INTEGER))
   (multislot resource-state (type SYMBOL))
   (multislot resource-setup (type SYMBOL)))
```

Listing 4.5: Definition of a usage-requirement of an event, in the event-based representation of the *scheduling model.* The usage-units of the requirement also determines wheter its *consumption-requirement* (i.e., units < 0), or a *production-requirement* (i.e., untis > 0).

Setup edge is a relation that exists between two events and a resource. It means that a resource *can* be used by the consumer event, directly after being released by a producer event. The duration needed between the events is the setup-duration. An event produces a resource at a specific resource-state.

```
(deftemplate setup-edge
   (slot sched-id (type SYMBOL))
   (slot resource-id (type SYMBOL))
   (slot producer-event (type SYMBOL))
   (slot consumer-event (type SYMBOL))
   (slot duration (type INTEGER))
   (multislot resource-state (type SYMBOL)))
```

Listing 4.6: Definition of a resource setup-edge between two events, in the event-based representation of the *scheduling model.* Setup durations are determined apriori.

Resource allocation the sequence of events served by a resource. *Formulated* for each resource with an initially empty *event-sequence.* Upon *expansion*, The sequence is populated with the allocation-schedule.

Schedule Life-cycle

A schedule instance goes through the stages of the scheduling process, in a *life-cycle* analogical to a goal life-cycle. The entire life-cycle of a schedule takes place during the expansion of a goal.

- **Formulated** when a goal of SCHEDULE-SUBGOALS sub-type and SCHEDULE *goal-class* is expanded. During formulation, sub-goals in the goal's sub-tree, the set of available resources, and solver parameters are specified. Upon formulation, the *event-based representation* is formulated for the specified goal and resource sets. After that, the schedule is selected.
- **Selected**, calling the solver to populate the datasets; generate the optimization model; triggers the optimization.
- **Expanded** when a solution is found (or infeasibility is proven)
- **Committed**, post-processing the goals; a plan is selected; resource-setup goals are formulated for each goal
- **Dispatched**, specifying the start-times of goals. This result is a *scheduled goal-tree*, ready to be executed.

Several schedule instances could be formulated and expanded for the same set of goals, before a schedule instance is committed. A schedule will only effect the goals, when the instance is *committed*.

Setup Durations

knowing the *setup states* (i.e., consumption-setup and production-setup) of each plan. An estimate is calculated of how long it would take to change from any *production-setup* to any *consumption-setup*. The *setup duration* needed by a resource to serve a plan a after being released be plan b (i.e., $a \prec b$, wrt the resource), is the duration needed to change the *setup predicate* from the *production-setup* of a to the *consumption-setup* of b. Estimates for all the *setup durations* is calculated apriori.

Lower-bound Calculation

Giving a lower-bounds for the variables of a minimization problem is employed by the *branch-bound* algorithm to speed-up the solution process. A lower-bound for a `goal-start` event,a lower-bound of 0, if no sub-goals exits; `plan-start` event has the same lower-bound as its `goal-start` event; `goal-end` event has a lower-bound equal its `plans-start` in addition to the duration of its shortest plan; `goal-start` event has a lower-bound equal to the highest lower-bound of its sub-goal's `goal-end` events.

Post Processing

For an *expanded* goal-tree of sub-type `SHCEDULE-SUBGOALS`, to start *committing* to goals, the *star-times* of goals has to be determined first. Several schedule instances could be *formulated*, *selected* and *expanded*. Yet, in order determine the *start-times* of goals, a single schedule has is *committed* and *dispatched*. Once a **schedule instance** is *committed*. The goal-tree is amended with the schedule execution-information. The following amendments are made to the expanded goal-tree, by post-processing the event-based representation of a schedule instance.

- **Plan selection** all non-scheduled plans are removed.
- **Plan resources** are elevated to become goal-resources. This is useful during execution; goals can only be *dispatched* after they had acquired their goal-resources.
- **Setup goals** are formulated for a resource upon post-processing of it's **resource-allocation**. A setup is needed when there is a (producer-event $\prec$ consumer-event) pair in the event-sequence which requires two different *setup states* (i.e., the **usage-requirement**s of the events has a different resource-setup).

The *scheduling model* serves as a communicator between the *solver* and the Clips-Executive. The scheduler uses the *event-based representation* to populate the data-sets needed to generate the MIP model. The scheduler also specifies any *solver parameters* needed by to influence the solving algorithms or alter the solution strategy (such as, setting a time limit).

4.5.3 MIP Model Generation

Upon fully formulating a schedule instance and its *event-based representation*, the MIP solver is called to populate the data-sets, used to generate the MIP model. Here we revisit the MIP model, giving more attention to the meaning of the date-sets and what does it correspond to from previous models. We use the notation $\widehat{=}$ to express a correspondence relation, **bold letters** to represent entities of the *event-based model* and sans sreif for their parameters. For the purpose of completion and easier viewing, the entire MIP model is shown below, fitted in a single page.

Data Sets

E	Set of atomic events $\widehat{=}$ (**Event**::event-names)
S	Set of selectors-names $\widehat{=}$ (**Event**::entities) (i.e., plans, goals or resources)
R	Set of resource-names $\widehat{=}$ (**Schedule**::resources)

Indices

$EC_r \subset E$	Subset of events which consumes a resource $r \in R$; $\widehat{=}$ **Event**::event-name of events which has a negative **Usage-Requirement**::units of r
$EP_r \subset E$	Subset of events which produce a resource $r \in R$; $\widehat{=}$ **Event**::event-name of events which has a positive **Usage-Requirement**::units of r
$SE_s \subset E$	Subset of events in a selection group, of selector $s \in S$; $\widehat{=}$ **Event**::event-names with the same **Event**::entity (i.e., events $\in$ same plan or the same goal)
$SG_s \subset S$	A select-one-of selection group of selectors, $s \in S$, where is $s \notin SG_s$; $\widehat{=}$ **Events**::entity of plans in a goal; $s \widehat{=}$ the goal (i.e., plans of a goal)

Parameters

M	A number larger than the upper bound, used to linearly model indicator constraints
$\alpha_{i,j} \in \{0,1\}$	wheter event $i \in E$ precedes event $j \in E$; $\widehat{=}$ **precedence**
δ_i	Duration of event $i \in E$; $\widehat{=}$ (**Events**::duration)
$\Delta_{r,i,j}$	Setup duration needed by resource $r \in R$ to serve event $i \in E_r^C$ directly after serving event $j \in E_r^C$
$\Theta_{r,i}$	Quantity required of $r \in R$ by event $i \in E$; $\widehat{=}$ (**Usage-Requirement**::units)
$\omega_s \in \{0,1\}$	wheter a selector $s \in S$ is initially set to true; $\widehat{=}$ (**Event**::scheduled)

Decision Variables

$X_{r,i,j} = \{0,1\}$	Resource $r \in R$ serves event $j \in E_r^C$ directly after serving event $i \in E_r^P$
$Z_s = \{0,1\}$	Selector $s \in S$ is selected
$T_e \in \mathbb{Z}$	Time of event $e \in E$
$T_{Max} \in \mathbb{Z}$	Completion time of all goals

Objective function: Minimize(T_{Max})

Constraints

$$\omega_s(1 - Z_s) \leq 0 \qquad \forall s \in S \tag{4.1}$$

$$\sum_{s \in SG_g} Z_s = Z_g \qquad \forall g \in S \tag{4.2}$$

$$\sum_{j \in EC_r} X_{rij} = Z_s \,.\Theta_{ri} \qquad \forall\ i \in EP_r\ \{i \in SE_s,\ r \in R,\ s \in S\} \tag{4.3}$$

$$\sum_{i \in EP_r} X_{rij} = Z_s \,.\Theta_{rj} \qquad \forall j \in EC_r\ \{j \in SE_s,\ r \in R,\ s \in S\} \tag{4.4}$$

$$\alpha_{i,j}\,(-T_j + T_i + \delta_i) \leq M(2 - Z_{si} - Z_{sj}) \qquad \forall i,j \in E\ \{i \in SE_{si},\ j \in SE_{sj}, si, sj \in S\} \tag{4.5}$$

$$-T_j + T_i + \delta_i + \Delta_{r,i,j} \leq M(1 - X_{rij}) \qquad \forall r \in R,\ \forall i \in EP_r,\ \forall j \in EC_r \tag{4.6}$$

$$T_i \leq T_{Max} \qquad \forall i \in E \tag{4.7}$$

Solution Processing

The optimization is performed asynchronously by the solver. The optimization status is checked periodically until an optimal solution has been is returned (or infeasibility has proven). Consequently, the solution of the MIP model, specifying the values of each decision variable, is processed and the corresponding *event-base representation* is updated. After processing the solution, the **schedule instance** mode is switched to *expanded.*

The decision variables are processed as follows:

- For each $T_e = x \in \mathbb{Z}$
  $\widehat{=}$ the **Event**::start-time is set to‘ x, for all events with **Event**::name = e.
- For each $Z_s = 1$,
  the **Event**::scheduled is set to True, for all events with **Event**::entity = s.

- For each $r \in R$ and all selected sequences $X_{r,e1,e2}, X_{r,e2,e3}, .., X_{r,e_{n-1},e_n} = 1$ $\widehat{=}$ the **Resource-Allocation**::event-sequence is set to $\{e_1, e_2, e_3, .., e_{n-1}, e_n\}$, for **Resource-Allocation**::resource $= r$.

4.5.4 Execution

A goal with sub-type `SCHEDULE-SUBGOALS` is *committed* when

- The goal start-time has been reached
- All sub-goals were completed
- It is the next goal allocated, for all of its goal-resources (i.e., no other goal is scheduled to use any of the goal-resources earlier than this goal)

Unpon commitment, a goal attempts to acquire the required goal-resources. Upon successful acquisition, the goal is *dispatched* and the execution of its plan is started.

This execution scheme is very useful in the unfortunate case that the temporal schedule is violated during execution (e.g., a goal took longer than expected, for instance if an MPS breaks; the estimate of a plan or a setup duration was inaccurate). Indeed, if a goal is delayed, it could at worst cause a delay in the execution of the goal-tree. The only case when the execution of a goal-tree would fail, if an action fails. In this case, it is necessary to start formulating a new goal-tree.

The sequence by which plans that directly depend on each other are executed, is insured by the sub-goal relations. For example, a C1 order `MOUNT-CAP` goal can only start after the `MOUNT-RING1` sub-tree and the `PREPARE-CAPSTATION` sub-tree have completed

The sequence by which plans which require the same resource are executed, is ensured by the resource allocation schedule. This ensures that a delay for a resource-allocation caused by a delayed goal in some sub-tree, effects the allocation of the goals using that same resource, in the other sub-trees. For example, a robot is allocated to `MOUNT-RING1` followed by `SETUP001-ROBOT`, `FILL-RING2`, `SETUP002-ROBOT`, `FILL-CAP` goals. If the robot was delayed on the first goal, we need to make sure it still starts next goals according to the resource-allocation sequence. Otherwise, the `FILL-RING2` goal will starve, or it could cause a threat that breaks a precondition (e.g., like `SETUP002-GOAL` moves the robot to a location unexpected by `FILL-RING2`).

Furthermore, the set of goal-resources is acquired (upon *committing*), only after we already had made sure that the goal is the next allocated for each resource in set. This prevents situations where a delayed goal attempts to acquire a resource, only to find that the resource had been acquired by another goal where its dispatch-time is started. This prevents deadlock situations where several goals compete for a acquiring a shared subset of resource, partially held by each of them.

Locking

Resource locking is performed using the mutex mechanism developed by , with the only adjustment that goals do not immediately fail when they're unable to acquire a goal-resource. A goal tries to acquire the goal-resources till it either succeeds, or the execution fails.

Remote Executors

Even though goal reasoning and scheduling are performed by the central reasoning agent, actions are only executable by a remote acting agent (e.g., a robot is responsible for executing its `move`, `pick-wp` actions). We extended the current CX agent to only `RUN` actions where the agent-name is mentioned as part of the PDDL action parameters.

An CX agent is stared on each robot. All agents (i.e., central-reasoner and remote-executors) synchronize their world model. Remote-executors are responsible for formulating their local maintenance goals (e.g., periodically sending the beacon-signal).

During execution, the central reasoner creates a `remote-action` world-model fact for `PENDING` actions which are not executable by the reasoner. The `remote-action` is synchronized with all agents, but only parsed into a `plan-action` by the proper agent (i.e., the agent which is mentioned in the action parameters). Consequently, the `plan-action` is executed normally by the remote agent, and the `remote-action` world-model fact is updated with the execution status. When the `EXECUTION-SUCCEEDS` the centralized-reasoner applies the effects. That achieves separation between reasoning and acting, while maintaining the remote agents ability to make local decisions. Indeed, reasoning about actions executability, preconditions and effect are performed by the central reasoner regardless of who performs the action.

Partially Instantiated Actions

Sometimes it is useful to decide as late as possible on the particular grounding of some action-parameters, leaving them open during goal-reasoning and scheduling. Such parameters could be a dynamic property of the environment, or parameter influenced by the particular plan ordering which the scheduler later determines.

For example, a `retrieve-cap` plan should specify which of the *cap-carriers* available at *each of the three shelf positions* of a cap-station is used to fill the cap-station. The *same* cap-carrier is the also used by the next `clear-cs` goal to empty the cap-station. Making such grounding decision to by the scheduler would unnecessarily increase the complexity of scheduling. Such a decision is only valuable during execution. Yet, it is still necessary to distinguish different cap-carriers used by two distinct `retrieve-cap` goals. Another prominent example is for the `fill-rs` and/or `mount-ring` goals which use the same ring-station's slide. Their plans should be instantiated specifying the number of bases added to the slide, thus far. Yet, there is no way to predicate this information before the scheduler had made its decision about the particular execution order of plans.

We devised a mechanism which allows us to specify a partial grounding of parameters during plan construction, leaving some binding decision to be later determined, when the action is selected for execution. The plans are instantiated using unique *binding-symbols*, created in relation to a *binding constraint*. When needed, a unification operator is invoked to determine the appropriate substitutions for the symbols, by matching the binding constraint to an existing world-model fact. The unification takes place right before an action is executed (i.e., action is selected in `PENDING` state).

For the `retrieve-cap` example, during plan construction a new binding-id `B#01` is used to create the binding constraint `(wp-on-shelf C-CS1 B#01?cc B#01?spot)`. The symbols `B#01?cc` and `B#01?spot` are used to instantiate the action-parameters. When the action is selected, the unification operator is invoked; the constraint `(wp-on-shelf C-CS1 B#01?cc B#01?spot)` is matched to the predicate `(wp-on-shelf C-CS1 CC-123 RIGHT)`, for instance; the binding-symbols are then substituted with the matched binding value (i.e., `CC-123` and `RIGHT`) before the action is executed.

A binding could also be used in the definition of other bindings. For the `fill-rs` example, the binding-ids `B#11` and `B#12` are used to create the constraints `(rs-filled-with C-RS1 B#11?rs-before)` and `(rs-inc B#11?rs-before B#12?rs-after)`. Unification of the binding `B#12` will trigger unification of binding `B#11`, substituting the symbol `B#11?rs-before` before matching `B#12`'s constraint.

Furthermore, we define two different *binding-policies* used in matching the *binding-constraints*, `MATCH-UNIQUE` and `MATCH-ANY`. Bindings using a `MATCH-UNIQUE` policy can only match against predicates that has not been used for another binding. Once a predicate is matched to a *binding-constraint*, it can not be match with a different one. On the other hand, `MATCH-ANY` policy can match several distinct *binding-constraints* with the same predicate.

For the `retrieve-cap` example, it is necessary to distinguish the different cap-carriers used by two distinct `retrieve-cap` goals. For that purpose the `MATCH-UNIQUE` policy is used to only ever use a cap-carrier once per binding. For the `fill-rs` example, we only care to know that state of the predicates at the time of execution. For that purpose the `MATCH-ANY` policy to allow several bindings to match against the same state predicate.

4.6 Evaluation

The approach was evaluated in simulation runs against the current Incremental Reasoning agent as a baseline. The simulation complies with the 2019 RCLL game [3] rules. We focus here on evaluating the efficiency improvement in single order production.

The performance metric considered consists of:

Total production duration needed by agents to competitively complete the production of the same order (makespan)

Scheduling duration and the effect of different order complexities on the scheduling time

Execution failure rate by breaking logical constraints during execution)

Schedule violations and its effects on execution times

• Approximately 80 games with several order complexities (i.e., C0, C1 , C2 , C3) and order configurations have been evaluated. The following was considered

- An average of 5 game runs per configuration.
- Configurations were run with different assumptions about robots speed. An average of 2.4 runs per configuration for a speed
- Estimates for travail durations are based on calculating the Euclidean distance between the navgraph nodes.
- More focus has been giving to evaluating orders which are more complex orders (C3) to produce; orders with different number of required sub-goals are considered

Figure 4.11 show production durations needed for the same configuration and speed. Our measures seems to be consistent.

4.6.1 Duration Estimates

The duration of machine operations were fixed based on means of real world observations. Robots speeds are configured based on rough estimates of the possible speeds of a robot in the real world. A maximum average speed of 0.5 meters per second was initially set. It was noticed that the baseline production speed greatly surpasses the real world observations. Indeed, when travailing duration is only a small factor of the total production duration, the production is almost entirely determined by durations of machine operations.

By evaluating different assumption for robot speeds, it was shown that the more crucial the travailing duration to the production process (i.e., the bigger the factor of production duration spent travailing), the bigger the delta by which our approach improves the baseline.

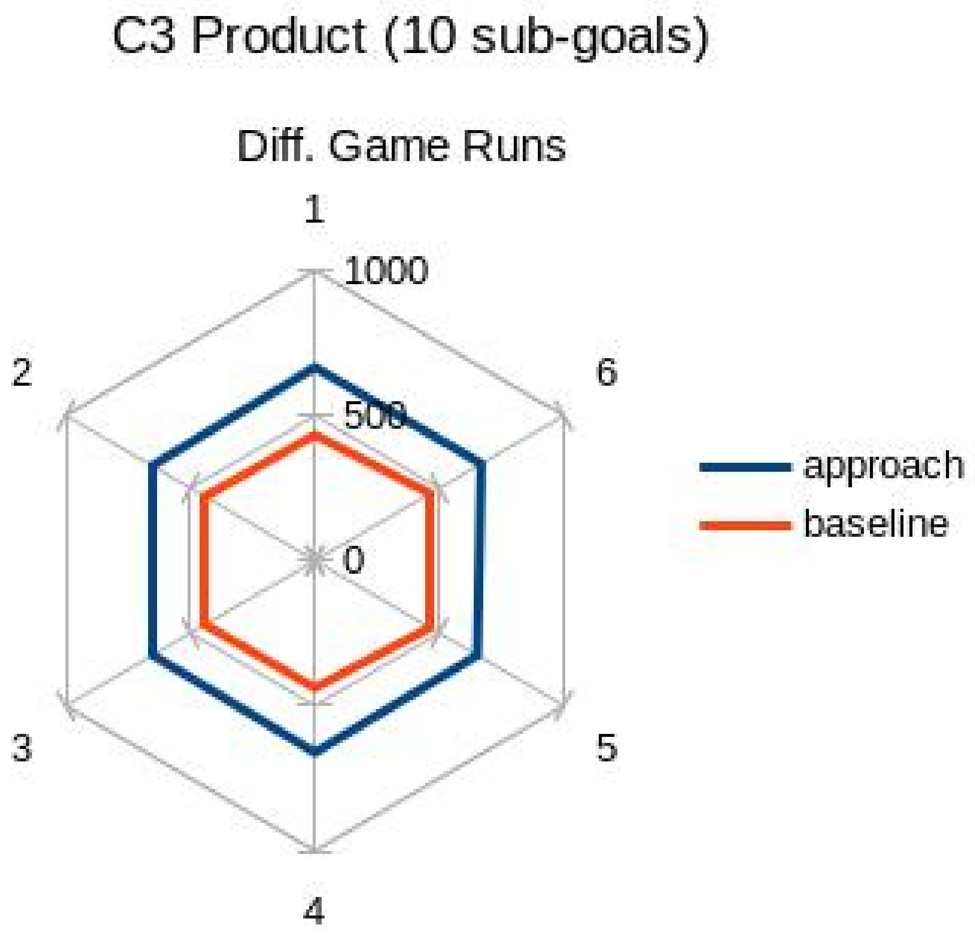


Figure 4.11: Five different simulation runs for the same production configuration.

The result in figure show the production times needed by an the agents for different speed assumption. Indeed, the optimistic speed factor of 0.5 m/sec puts the incremental reasoning at a great advantage, since the time penalty incurred as a result of picking a sub optimal goal is minimized. The incremental agent picks a the best possible goal at the time of reasoning without considering or expecting future resource availability. When travailing times are small, the best available goal is indeed the optimal goal.

On the other hand, a more realistic estimate incorporates a bigger factor of production as travail durations. Observations of real-world games show that the biggest factor of production time spent by robots is during travailing. Out approach is evaluated against the baseline in a game run with different (common) speed assumption. Figure 4.12 show the run times of a C3 order configuration where robots executes 9 sub-goal to deliver an order. An optimistic, a pessimistic, and a midrange speed assumptions where evaluated. Our approach out performs the baseline when complicated production sequences are required, and when travailing time constitutes a big factor of production time.

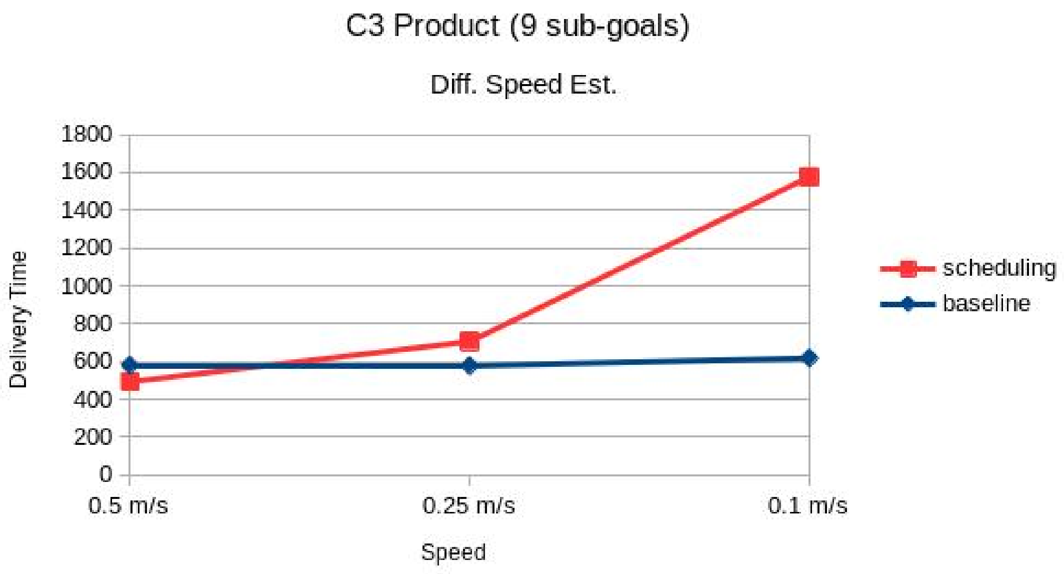


Figure 4.12: A C3 configuration run with different speed assumptions

One other reason why our near optimal approach performs poorly in the optimistic assumption is a observed systematic schedule delay.

4.6.2 Execution Delay

During evaluation an execution delay of an average of 60 seconds (from schedule) was observed. The delay is attributed to a factor of the number of actions in a goal. By rolling out all causes of delay, like action execution durations or poor action estimates, the delay was found to be a factor of plan length. We use this delay to out advantage to reflect potential real world schedule violations.

The systematic delay as well as some induced poor duration estimates were used to evaluate the stability of our production against uncertainty. It was observed that our centralized reasoning and scheduling approach does not break as a result of poor of schedule violation. Our execution scheme insures that execution will continue seamlessly, with a schedule delay. This is a result of using the allocation schedule for each resource as well as the execution schedule.

4.6.3 Optimality and optimality gap

It was observed that most of the optimization time spent by the solver, is close enough from proving optimality. We use our lower-bound estimates to give a close enough estimate of the minimum production time. An optimality gap of 10% is evaluated in our approach. Figure 4.13 shows that the execution time is barley affected, while the scheduling time is reduced by an average of 95

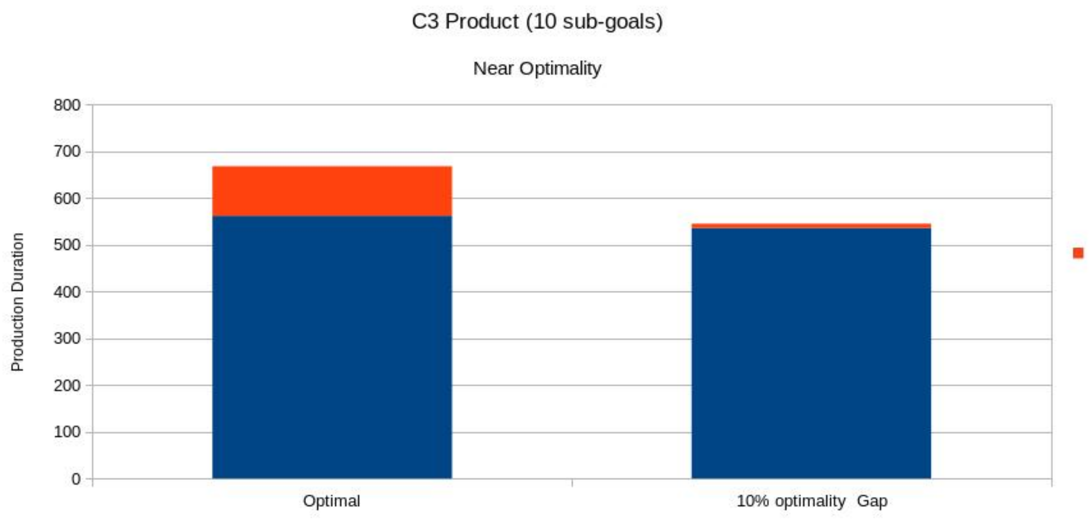


Figure 4.13: A C3 configuration run with different speed assumptions

4.6.4 Different orders

We evaluated our approach by running different configurations of each order (i.e., C0, C1 , C2 and C3) to get an overview on how much does our approach improve the production of each. Figures **??** and 4.14 shows the different attempted order configurations, evaluated for the max and midrange speed assumptions.

Worthy of noting that a C1 product ($C1_B$), which requires the payment of two raw material the baseline performs very poorly. This is due that lack of favouring order oriented goals (i.e., lack of deliberation of a common goal), which renders other available goals more favourably. This could be easily remedied by changing the goal selection strategy to favour ring payments tasks of open C1 products.

It is observable that in the case of a C0 our approach does has no noticeable improvements, indeed for a C0 the production sequence is fixed and the incremental agent as well as the central agent execute the same sequence. When no complicated interaction and scheduling is needed the incremental agent performs better. Nevertheless, when complicated production sequence is needed, our approach prevails.

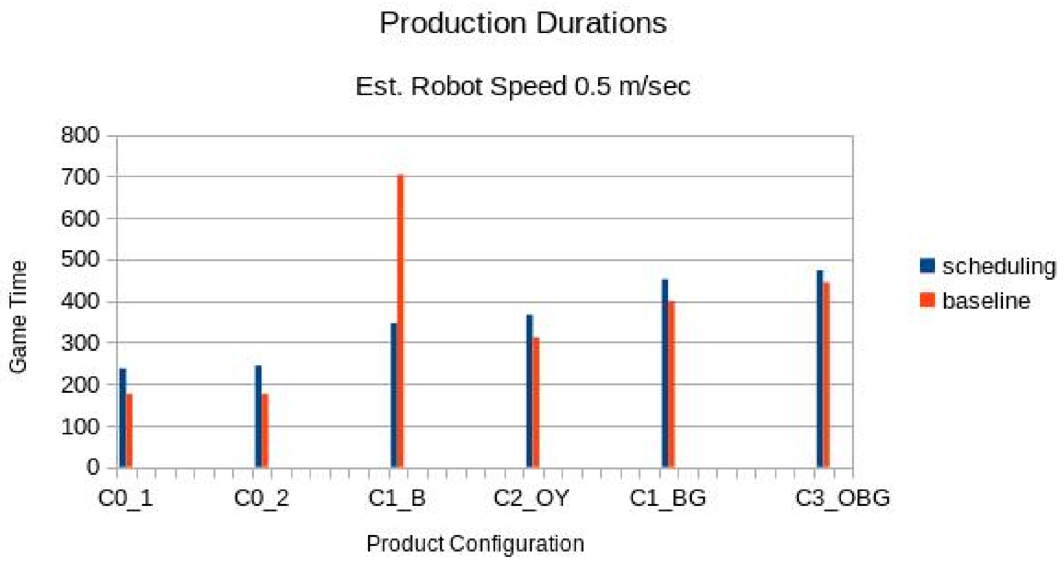


Figure 4.14: Different orders evaluated with the .5 msec robot speeds

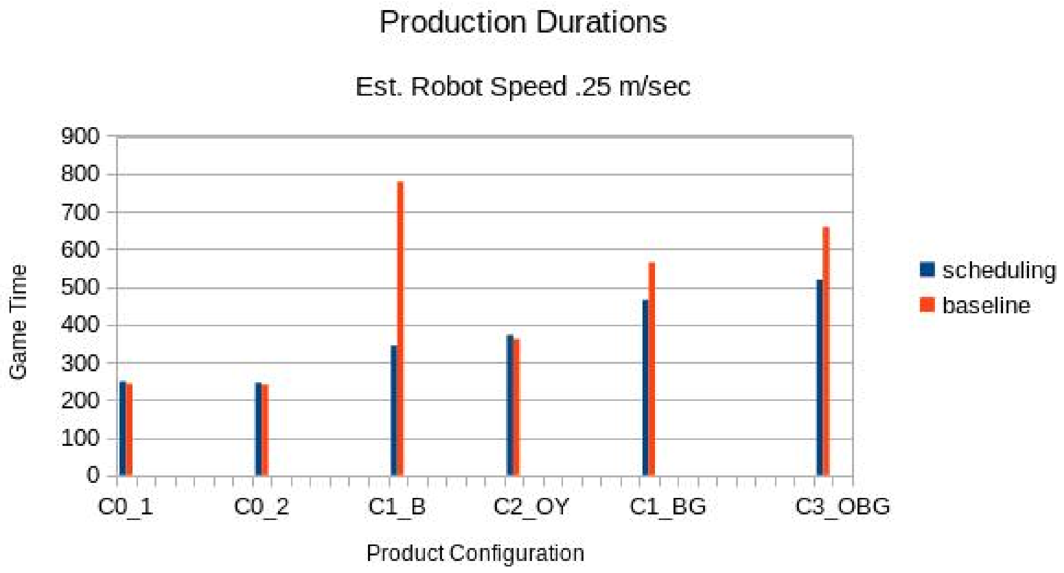


Figure 4.15: Different orders evaluated with the .25 msec robot speeds

4.7 Conclusion

In order to improve production time of the current incremental goal reasoning maintained by the Carologistics team, the world champion of the RCLL, a centralized goal reasoning and scheduling approach was developed.

This thesis described the development and integration of a scheduling model, with in the centralized goal reasoning model. We presented a MIP formulation, dynamically generated by our scheduling model to solve the order scheduling problem to optimality. To abide by the real-time requirement, it was shown that using a near optimal solution, with an optimality gap of 10%, reduces the order scheduling time by 95%.

Our approach is evaluated in a number of simulation runs against the incremental agent. It was shown that the more the production time spent by robot on travailing, the more surpassing our approach becomes. Our approach also out performs the baseline when complex production scenarios are required.

Bibliography

[1] David W. Aha. Goal reasoning: Foundations, emerging applications, and prospects. *AI Magazine*, 39(2):3–24, Jul. 2018. doi: 10.1609/aimag.v39i2.2800. URL `https://www.aaai.org/ojs/index.php/aimagazine/article/view/2800`.

[2] E.M.L Beale. *Mathmatical Programming in Practice*. Pitman, 1968.

[3] Coelen, Deppe, Gomaa, Hofmann, Karras, Niemueller, Rohr, and Ulz. RoboCup Logistics League Rules and Regulations 2019 The Technical Committee 2012 – 2019. Technical report, 2019. URL `https://github.com/robocup-logistics/rcll-rulebook/releases/download/2019/rulebook2019.pdf`.

[4] William Cushing, Subbarao Kambhampati, Mausam, and Daniel S Weld. When is temporal planning {\em really} temporal planning? *Proc. of Int. Joint Conf. on AI (IJCAI)*, pages 1852–1859, 2007.

[5] George B. Dantzig. *Linear programming and extensions*. Rand Corporation Research Study. Princeton Univ. Press, Princeton, NJ, 1963. URL `http://gso.gbv.de/DB=2.1/CMD?ACT=SRCHA&SRT=YOP&IKT=1016&TRM=ppn+180926950&sourceid=fbw_bibsonomy`.

[6] Maria Fox and Derek Long. PDDL2.1: An extension to PDDL for expressing temporal planning domains. *Journal of Artificial Intelligence Research (JAIR)*, 20, 2003. doi: 10.1613/jair.1129.

[7] Malik Ghallab, Dana Nau, and Paolo Traverso. *Automated planning - theory and practice*. Morgan Kaufmann Publishers, 2004.

[8] R L Graham, E L Lawler, J K Lenstra, and A H G R Kan. Optimization and approximation in deterministic machine scheduling: a survey. *Annals of Discrete Mathematics*, 5:287–326, 1979.

[9] Gurobi-Optimizer. OR-framwork, 2019. URL `https://www.gurobi.com/documentation/8.0/refman/index.html`.

[10] Keith Halsey, Derek Long, and Maria Fox. CRIKEY - A Temporal Planner Looking at the Integration of Scheduling and Planning. *Workshop on Integrating Planning into Scheduling*, 02:46, 2004.

[11] Mario Hermann, Tobias Pentek, and Boris Otto. Design principles for industrie 4.0 scenarios. *Proceedings of the Annual Hawaii International Conference on System*

Sciences, 2016-March:3928–3937, 2016. ISSN 15301605. doi: 10.1109/HICSS.2016.488.

[12] Till Hofmann, Nicolas Limpert, Victor Matar, Alexander Ferrein, and Gerhard Lakemeyer. Winning the RoboCup Logistics League with Fast Navigation , Precise Manipulation , and Robust Goal Reasoning. *RobotCup*, XXIII, 2019.

[13] Henning Kagermann, Wolfgang Wahlster, and Johannes Helbig. Recommendations for implementing the strategic initiative industrie 4.0 – securing the future of german manufacturing industry. Final report of the industrie 4.0 working group. URL `http://forschungsunion.de/pdf/industrie_4_0_final_report.pdf`.

[14] Klara L. Hoffman and Ted K. Ralphs. Integer and Combinatorial Optimization. *Encyclopedia of Operations Research and Manage-ment Scienc*, pages 771–783, 2013. ISSN 01605682. doi: 10.2307/2583737.

[15] A. H. Land and A. G. Doig. An Automatic Method of Solving Discrete Programming Problems. *Econometrica*, 28(3):497–520, 1960.

[16] Francesco Leofante, Erika Abraham, Tim Niemueller, Gerhard Lakemeyer, and Armando Tacchella. Integrated Synthesis and Execution of Optimal Plans for Multi-Robot Systems in Logistics. *Information Systems Frontiers*, pages 1–21, may 2018. ISSN 1572-9419. doi: 10.1007/s10796-018-9858-3. URL `https://doi.org/10.1007/s10796-018-9858-3`.

[17] Matthias Löbach. *Centralized Global Task Planning with Temporal Aspects on a Group of Mobile Robots in the RoboCup Logistics League.* Master's thesis (to appear), RWTH Aachen University, 2017.

[18] Lars Lütjens, Martin Bichler, Stefan Minner, and Albinski Syzmon. Task Allocation for Fleets of Mobile Autonomous Picking Robots, 2018. ISSN 1098-6596.

[19] Drew McDermott. The Formal Semantics of Processes in PDDL. In *Proceedings of the Workshop on PDDL at the 13th International Conference on Automated Planning & Scheduling (ICAPS)*, 2003.

[20] Merriam-Webster. OR, 2019. URL `https://www.merriam-webster.com/dictionary/operations%20research`.

[21] Hans Mittelmann. OR, 2019. URL `http://plato.asu.edu/ftp/lpcom.html`.

[22] Hans Mittelmann. OR, 2019. URL `http://plato.asu.edu/ftp/milpf.html`.

[23] George L. Nemhauser and Laurence A. Wolsey. *Integer and Combinatorial Optimization.* Wiley-Interscience, New York, NY, USA, 1988. ISBN 0-471-82819-X.

[24] Tim Niemueller, Gerhard Lakemeyer, and Alexander Ferrein. Incremental task-level reasoning in a competitive factory automation scenario. In *AAAI Spring Symposium: Designing Intelligent Robots*, 2013.

[25] Tim Niemueller, Till Hofmann, and Gerhard Lakemeyer. Goal reasoning in the CLIPS executive for integrated planning and execution. In *Proceedings of the Twenty-Ninth International Conference on Automated Planning and Scheduling, ICAPS 2018, Berkeley, CA, USA, July 11-15, 2019.*, pages 754–763, 2019. URL `https://aaai.org/ojs/index.php/ICAPS/article/view/3544`.

[26] Ngutor Nyor, Adamu Idama, J.O. Omolehin, and K. Rauf. Operations Research-What It is all About. *Universal Journal of Applied Science*, 2(3):57–63, 2014. doi: 10.13189/ujas.2014.020301. URL `http://www.hrpub.org`.

[27] Jason Chao Hsien Pan and Jen Shiang Chen. Mixed binary integer programming formulations for the reentrant job shop scheduling problem. *Computers and Operations Research*, 32(5):1197–1212, 2005. ISSN 03050548. doi: 10.1016/j.cor.2003.10.004.

[28] Mark Roberts, Swaroop Vattam, Ronald Alford, Bryan Auslander, Justin Karneeb, Matthew Molineaux, Tom Apker, Mark Wilson, James McMahon, and David W. Aha. Iterative goal refinement for robotics. In *A. Finzi & A. Orlandini (Eds.) Planning and Robotics: Papers from the ICAPS Workshop*, Portsmouth, NH, USA, 2014. URL `http://www.nrl.navy.mil/itd/aic/sites/www.nrl.navy.mil.itd.aic/files/pdfs/Roberts-2014-ICAPS-WS.pdf`.

[29] Rjm Rob Vaessens. *Generalized job shop scheduling : complexity and local Generalized Job Shop Scheduling : Complexity and Local Search.* PhD thesis, Universiteit Technische Doi, Eindhoven, Netherlands, 1995.

[30] Thomas Voß, Jens Heger, Nicolas Meier, and Anthimos Georgiads. Optimal robot scheduling of an AGV in the RCLL and introduction of team Leuphana. 2016.

[31] Robert M. Wygant. CLIPS — A powerful development and delivery expert system tool. *Computers & Industrial Engineering*, 17(1-4), 1989. doi: 10.1016/0360-8352(89)90121-6.