SCHEDULING THROUGH LOGIC-BASED TOOLS

Jordi Coll Caballero

DOCTORAL THESIS

Scheduling Through Logic-Based Tools

Author:
Jordi Coll Caballero

2019

DOCTORAL THESIS

Scheduling Through Logic-Based Tools

Author:
Jordi Coll Caballero

2019

PhD Program in Technology

Supervisors:
Dr. Josep Suy Franch
Dr. Mateu Villaret Auselló

Thesis submitted in fulfillment of the requirements for the degree of Doctor of Philosophy by the University of Girona

Universitat de Girona

Dr. Josep Suy Franch, of Universitat de Girona, and Dr. Mateu Villaret Ausellé, of Universitat de Girona,

WE DECLARE:

That the thesis titled *Scheduling Through Logic-Based Tools*, presented by Jordi Coll Caballero to obtain a doctoral degree, has been completed under our supervision and meets the requirements to opt for an International Doctorate.

For all intents and purposes, we hereby sign this document.

Girona, 9th of April of 2019.

Acknowledgements

First of all I want to express my most sincere gratitude to the Logic and Programming research group of Universitat de Girona, where I have belonged as a PhD student. To my supervisors Josep Suy and Mateu Villaret, who have accompanied me during all this project, who have put their best efforts in helping me grow professionally and personally, and from whom I have learned invaluable lessons. I also want to extend this gratitude to Miquel Bofill, with whom I have spent many hours working side by side. And also to my laboratory fellows Joan Espasa and Gerard Martín, with whom I have had really good times.

I would also like to thank the IMAE department of Universitat de Girona, where I have found the best of the environments and colleagues to work during these years.

A very deep gratitude to all the members of the constraints group at the University of St. Andrews, who have very gently accepted me as one of them during my visit. A especial acknowledgement is for Ian Miguel, Peter Nightingale and András Salamon, with whom I have been closely working and who have helped me in multiple occasions.

And last but not least, I want to thank my friends and family, especially my partner Rosa and my parents Quim and Paqui, whose unconditional support has been indispensable to move forward on this thesis. If it were not for them, I would have not been able to stand where I am.

This thesis has been supported by grant TIN2015-66293-R (MINECO / FEDER, UE), grant RTI2018-095609-B-I00, grant MPCUdG2016/055 (UdG), grant IFUdG2016 (UdG), and *Ayudas para Contratos Predoctorales 2016* (grant number BES-2016-076867, funded by MINECO and co-funded by FSE).

Publications

- Conference Proceedings derived from this thesis:
  - Miquel Bofill, Jordi Coll, Josep Suy and Mateu Villaret, *Solving the Multi-Mode Resource-Constrained Project Scheduling Problem with SMT*. IEEE International Conference on Tools with Artificial Intelligence (ICTAI), 2016.
    CORE: B, GSS Conference Rating: B
  - Miquel Bofill, Jordi Coll, Josep Suy and Mateu Villaret, *Compact MDDs for Pseudo-Boolean Constraints with At-Most-One Relations in Resource-Constrained Scheduling Problems*. International Joint Conference on Artificial Intelligence (IJCAI), 2017.
    CORE: A++, GSS Conference Rating: A++
  - Miquel Bofill, Jordi Coll, Josep Suy and Mateu Villaret, *An Efficient SMT Approach to Solve MRCPSP/max Instances with Tight Constraints on Resources*. Principles and Practice of Constraint Programming (CP), 2017.
    CORE: A, GSS Conference Rating: A
  - Miquel Bofill, Jordi Coll, Josep Suy and Mateu Villaret, *SAT Encodings of Pseudo-Boolean Constraints with At-Most-One Relations*. International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research (CPAIOR), 2019.
    CORE: B, GSS Conference Rating: B-
- Under review at the time that this thesis have been submitted:
  - Miquel Bofill, Jordi Coll, Josep Suy and Mateu Villaret, *An MDD-based SAT Encoding for Pseudo-Boolean Constraints with At-Most-One Relations*. Artificial Intelligence Review, 2019.
    Impact factor: 3.814, Ranking: Q1

List of Figures

List of Tables

List of Algorithms

Contents

Resum

Un problema de *scheduling* (programació de tasques), consisteix en decidir quan i com s'han d'executar les activitats d'un projecte, per tal de satisfer un seguit de requeriments. Avui en dia aquests problemes són molt presents en els sectors de la industria i els serveis. En la majoria de casos, trobar una solució d'un problema de scheduling és costós, especialment quan s'optimitza algun objectiu com ara la durada del projecte. Els avenços recents en solucionar els problemes de la Satisfactibilitat Booleana (SAT) i SAT Mòdul Teories (SMT) han despertat interès en formular problemes combinatoris durs com a fórmules SAT o SMT, que són resoltes amb algorismes eficients. Un dels principals avantatges d'aquestes tècniques basades en la lògica és que poden certificar solucions òptimes.

La contribució principal d'aquesta tesi és presentar mètodes eficients basats en SMT per solucionar problemes de scheduling. Concretament, ataquem el conegut *Resource-Constrained Project Scheduling Problem* (RCPSP), així com diverses extensions d'aquest que presenten requeriments i reptes addicionals. Aquestes extensions són les abastament conegudes: MRCPSP, RCPSP/t, MRCPSP/max i MSPSP. Les restriccions que presenten un major repte en problemes tipus RCPSP són les d'ús de recursos finits, normalment renovables, que són compartits entre activitats. Per tal de tractar aquestes restriccions, utilitzem codificacions a SAT de restriccions pseudo-Booleanes (PB), basades en diagrames de decisió. Com que aquestes codificacions tenen un gran impacte en els temps de solució, aprofitem restriccions col·laterals per codificar restriccions PB de manera compacta, tot preservant bones propietats de propagació. Tanmateix anem un pas més enllà, perquè creiem que aquesta tècnica per codificar PBs pot ser útil en àmbits diferents de scheduling. Amb aquest propòsit, dissenyem les nostres codificacions de PB de manera independent d'on s'apliquen, i proporcionem alternatives als diagrames de decisió. Les eines que presentem superen en rendiment les millors eines exactes existents per solucionar els problemes de scheduling estudiats.

Resumen

Un problema de *scheduling* (programación de tareas) consiste en decidir cuando y cómo deben ejecutarse las actividades de un proyecto, con el fin de satisfacer una lista de requerimientos. Hoy en día estos problemas son muy presentes en sectores como la industria y los servicios. En la mayoría de los casos, encontrar una solución de un problema de scheduling es costoso, especialmente cuando se optimiza algún objetivo como la duración del proyecto. Las mejoras recientes en solucionar los problemas de la Satisfacibilitad Booleana (SAT) y SAT Módulo Teorías (SMT) han suscitado interés en formular problemas combinatorios difíciles mediante fórmulas SAT o SMT, que son resueltas con algoritmos eficientes. Una de las principales ventajas de estas técnicas lógicas es que pueden certificar soluciones óptimas.

La contribución principal de esta tesis es presentar métodos eficientes basados en SMT para solucionar problemas de scheduling. Concretamente, atacamos el conocido *Resource-Constrained Project Scheduling Problem* (RCPSP), así como extensiones de este que presentan requerimientos y retos adicionales. Estas extensiones son las ampliamente conocidas: MRCPSP, RCPSP/t, MRCPSP/max y MSPSP. Las restricciones que presentan un mayor reto en problemas tipo RCPSP son las de uso de recursos finitos, normalmente renovables, que son compartidos entre actividades. Para tratar estas restricciones, utilizamos codificaciones a SAT de restricciones pseudo-Booleanas (PB), basadas en diagramas de decisión. Ya que estas codificaciones tienen un gran impacto en el tiempo de solución, aprovechamos restricciones colaterales para codificar restricciones PB de manera compacta, mientras preservamos buenas propiedades de propagación. Sin embargo, vamos un paso más allá, pues creemos que dicha técnica para codificar PBs puede ser útil en ámbitos distintos de scheduling. Con este propósito, diseñamos nuestras codificaciones de PB de manera independendiente de dónde se aplican, y proporcionamos alternativas a los diagramas de decisión. Los sistemas que presentamos superan en rendimiento a los mejores sistemas exactos existentes para solucionar los problemas de scheduling estudiados.

Abstract

A scheduling problem can be defined in a nutshell as the problem of determining when and how the activities of a project have to be run, according to some project requirements. Such problems are ubiquitous nowadays since they frequently appear in industry and services. In most cases the computation of solutions of scheduling problems is hard, especially when some objective, such as the duration of the project, has to be optimised. The recent performance advances on solving the problems of Boolean Satisfiability (SAT) and SAT Modulo Theories (SMT) have risen the interest in formulating hard combinatorial problems as SAT or SMT formulas, which are then solved with efficient algorithms. One of the principal advantages of such logic-based techniques is that they can certify optimality of solutions.

The main contribution of this thesis is the presentation of efficient SMT-based methods to solve scheduling problems. More precisely we tackle the well-known Resource-Constrained Project Scheduling Problem (RCPSP) as well as many extensions of this problem with additional requirements and modelling challenges. Namely, we also solve the problems commonly denoted by MRCPSP, RCPSP/t, MRCPSP/max and MSPSP. The most challenging constraints in RCPSP-based problems are resource constraints, which specify a limited availability of shared resources, usually renewable, that activities cannot surpass at any time. To handle such constraints we use decision diagram based SAT encodings of pseudo-Boolean (PB) constraints. Since these encodings have a high impact on solving times, we take advantage of collateral constraints to compactly encode PB constraints, while preserving good propagation properties. However we go one step further, because we believe that such PB encoding technique can be useful in other fields different than scheduling. With this idea in mind, we design our PB encodings in an application-independent way, and we provide many encoding alternatives different from decision diagram representations. The systems that we present are able to outperform the best state-of-the-art exact solvers for the studied scheduling problems.

Chapter 1

Introduction

1.1 Motivation

Hard combinatorial problems are ubiquitous nowadays. Such problems are usually modelled as a set of variables with domains and some constraint over them. These models are sometimes called *Constraint Satisfaction Problems* (CSP), and solving them consists in finding values for their variables satisfying a set of restrictions. Also it is often required to optimise an objective function, in which case they are referred to as *Constrained Optimisation Problems*. Some of the iconic CSPs that are frequently used by way of example for academic purposes include the *N-queens puzzle*, *sudoku* or logic enigmas such as *Einstein's problem*. However, CSPs appear in a large range of domains and have a myriad of applications in industry and services. Some examples are: *routing* problems, where it has to be decided which path has to follow an item or a set of items to reach their destination, for instance vehicle traffic control or data network management; *packing* problems, where a set of items have to be packed usually maximising a global profit, for instance loading of merchandise or server allocation; *timetabling* problems, like academic calendars or business meetings organisation.

Among CSPs we can find *scheduling* problems, that can be defined in a nutshell as the problem of determining when and how the activities of a project have to be run, according to some project requirements and limitations. The most frequent constraints involve precedence relations between the execution of the activities and correct allocation of shared resources with limited availabilities. Scheduling problems usually require to optimise some value, for instance minimise the total duration of the project. An example problem is to find a schedule for a manufacturing process. In this case there would be a set of activities, each one consisting in making an item, either from raw material or by

assembling other items. We cannot make an item until all its components have already been made, hence a precedence relation appears. Also we can have renewable resources, e.g. there is a limited number of workers and machines which can only work in one item at a time, and non-renewable resources, e.g. a limited amount of raw material or a budget.

Similarly to most CSPs, scheduling problems are usually NP-hard and therefore finding a solution for them may be practically impossible for fairly large instances —unless P=NP. Therefore, large scheduling problems require approximate techniques to find as good as possible solutions within an acceptable computation time for the particular application at hand. However, such approximate methods are not able to certify the optimality of the solutions. Nevertheless, exact solving methods —i.e. those which can find and certify the optimality of the solutions— have evolved in the past years and can be very efficient at solving problem instances with sizes of practical interest. One particularly interesting approach to use these methods is *model-and-solve*, consisting in specifying the problem at hand in a formal language, and using a specialised solver to find a solution for this specification. This way of solving CSPs brings the worlds of formal definition and solver development closer, and there are many formalisms which provide efficient CSP solving. Among the most studied we can find Mixed Integer Linear Programming (MILP), Constraint Programming (CP), Answer Set Programming (ASP), the problem of Boolean Satisfiability (SAT), or its extension Satisfiability Modulo Theories (SMT).

In particular, SAT solvers have improved dramatically during the last two decades, with the inclusion of learning techniques in the search process [MSS99] and very efficient implementations and search guidance heuristics [MMZ+01]. SMT solvers also benefit from these efficient techniques, since they use a SAT solver as a core component. Moreover, SMT has the additional advantage of being a very expressive language, supporting other kinds of logics such as linear arithmetic or uninterpreted functions with equality. SMT solvers use efficient and specialised solvers to handle these logics. Therefore, SMT provides a good compromise between efficiency and expressivity.

Due to the good evolution of SAT and SMT there is a growing interest in exploring these to model and solve CSPs [BPSV12]. The application of SMT to solve scheduling problems has already shown to be a very efficient approach [Suy13]. This thesis aims at pushing further in the efficiency of exact solving of hard scheduling problems by means of using the logic-based techniques, namely we will focus on the model-and-solve approach using SAT and SMT.

1.2 Objectives and Contributions

The goal of this thesis is to develop logic-based exact solvers to efficiently solve scheduling problems. More precisely, our objectives are the following:

1. We want to provide SMT formulations for many scheduling problems. SMT provides an expressive language as well as efficiency in solving formulas. Therefore, we think that this formalism is very suitable to express an solve the different kinds of constraints that are present in scheduling problems.

2. We want to explore the use of decision diagram based SAT encodings for resource constraints. We are aware of previous works that report good results in SAT encodings of pseudo-Boolean constraints, and this technique can be applied to resource constraints.

3. We want to study a new generic framework to encode pseudo-Boolean constraints taking into account collateral constraints of the problem at hand. We believe that this technique can be interesting by itself and have application not only in scheduling but in many CSPs. However, we want to deepen the application of this technique to scheduling problems, in particular in resource constraints.

4. We want to explore different types of encodings where we can take into account collateral constraints when encoding pseudo-Boolean constraints to SAT.

5. We aim at providing scheduling solvers which are competitive with the state-of-the-art exact methods.

1.3 Outline of the Thesis

In the rest of this document, we start by introducing in Chapter 2 some preliminaries on the areas covered by this thesis. First, we include an overview on types of scheduling problems, and introduce the well-known preprocessing and modelling techniques which we will be using. After that we provide a gentle overview on SAT and SMT solving, with basic concepts and notation. Finally, we review basic concepts on Binary Decision Diagrams (BDDs), their application to represent PB constraints, and different existing encodings of them to SAT.

The first goal that we have tackled in this thesis is using SMT to solve the Multi-mode Resource-Constrained Project Scheduling Problem (MRCPSP).

This is a well-known scheduling problem which presents many interesting constraints, such as both renewable and non-renewable resource constraints, and the appropriate selection of execution modes. This problem had already been solved with SMT [Suy13], but there were many improvement opportunities that we wanted to explore, such as heuristic computation of upper bounds, and the use of SAT encodings of Binary Decision Diagrams (BDD) to deal with resource constraints. Our results situate SMT as the state-of-the-art in solving the MRCPSP. This initial work is presented in Chapter 3, and it was published in [BCSV16].

The good results obtained with BDD encodings have encouraged us to push further in this direction, since we saw the opportunity of making the encodings much smaller by obtaining more compact decision diagram representations of the constraints. We noticed that there appear many notions of incompatibility in scheduling problems, and that these can be used to get more compact encodings of resource constraints. We developed an encoding for pseudo-Boolean (PB) constraints which appear in conjunction with some at-most-one (AMO) constraints. This technique let us obtain compact encodings of resource constraints, and efficiently solve extensions of the Resource-Constrained Project Scheduling Problem (RCPSP). These contributions were initially published in [BCSV17b], and we have extended them in Chapters 4, 5 and 6.

Chapter 4 is devoted to a deep analysis about how to encode of PB constraints which appear in conjunction with AMO constraints. We present generic techniques that can have application to other domains different than scheduling. In particular we introduce the idea of PB($\mathcal{C}$) constraints as a way of compactly encoding pseudo-Boolean (PB) constraints —like resource constraints— to SAT, taking into account collateral information of the problem at hand. We focus on PB(AMO) constraints, which apply in scenarios where PB constraints appear in conjunction with AMO constraints. We propose a Multivalued Decision Diagram (MDD) based encoding for PB(AMO) constraints, study its properties, and empirically evaluate its efficiency. With this new encoding technique, we are able to generate dramatically smaller SAT formulas, and the solving times improve remarkably. We remark that the contribution of this chapter is application-independent, since we provide methods to efficiently encode arithmetic constraints to SAT, and these methods can have a highly beneficial effect on SAT modelling and reformulation.

In Chapter 5 we describe the integration of the generic PB(AMO) encoding technique of Chapter 4 to SMT formulations of scheduling problems. We focus on the RCPSP, since it is the most basic scheduling problem. We provide efficient methods to detect incompatibilities between activities, and from there infer AMO constraints. Using this information we provide an SMT formulation

of the RCPSP which not only proves to be very efficient, but also is a basis upon which to design formulations for many extensions of the RCPSP.

In Chapter 6 we tackle more challenging scheduling problems using the methods presented in Chapter 5. We revisit the MRCPSP, already studied in Chapter 3, to provide a better formulation, and we also consider other extensions: variability of resource demands and availabilities in different time instants (RCPSP/t), generalised precedence relations with minimal and maximal time lags (MRCPSP/max), and assignment of resources which can perform different skills (MSPSP). We compare our solvers with the best state-of-the-art exact systems for each one of the problems, and we prove to be the best alternative on most occasions. The contributions for MRCPSP and RCPSP/t are published in [BCSV17b], and the ones of MRCPSP/max in [BCSV17a].

In Chapter 7 we revisit PB(AMO) constraints. There exist many different SAT encoding approaches for PB constraints, different than BDD-based ones, which have given good results. In this chapter we generalise them to SAT encodings of PB(AMO) constraints. The experimental section reports good results in application-independent instances, showing the potential of this approach that could enhance SAT solving of many problem classes. These contributions are published in [BCSV19].

In Chapter 8 we present the conclusions of the thesis, and we discuss some further work that could give a continuation to our contributions.

Chapter 2

Preliminaries

In this chapter we introduce some preliminaries on the areas covered by this thesis. Section 2.1 is devoted to scheduling problems. We include an overview of types of scheduling problems, and introduce the well-known preprocessing and modelling techniques which we will be using. In Section 2.2 we provide a gentle overview of SAT and SMT solving, with basic concepts and notation. Finally, in Section 2.3 we review basic concepts of Binary Decision Diagrams (BDDs), their application to represent PB constraints, and different existing encodings of BDDs to SAT.

2.1 Scheduling

This section contains a review of different existing classes of scheduling problems, some solving approaches, and introduces some notation and preliminaries related to the thesis. For an extensive review of scheduling problems and solving techniques the reader may refer to [ADN13].

2.1.1 A Review of Scheduling Problems

The identification of different kinds of scheduling problems and the pursue of efficient algorithms to solve them is a very active research field that has been drawing the attention of researchers since the 1960s. One of the first surveys on scheduling problems was [Dav73], that distinguished three main classes of scheduling problems: problems in which there is a trade-off between the time dedicated to a project and the cost it represents, problems in which resource demands are levelled, and problems with fixed resource limits. The latter type of problems is typically represented by the paradigmatic *Resource-Constrained Project Scheduling Problem* (RCPSP), which plays a central role in this thesis.

The goal of RCPSP is to find a start time for each one of the activities of a project such that the makespan is minimised. The makespan is defined as the total duration of the project. The activities have a predefined duration and are non-preemptive, i.e. they have to be executed without interruptions. Also, there is a set of requested end-start precedences between pairs of activities that have to be respected, meaning that the successor cannot start until the predecessor has ended. Finally, there is a set of renewable resources with finite capacity, and every activity has a certain demand on each resource while it is running. The total demand on a resource at any particular time cannot be greater than the resource capacity. In this thesis we mainly tackle scheduling problems which extend the RCPSP, and therefore Section 2.1.2 is devoted to introducing this problem in detail.

The constraints that appear in RCPSP are common in a large variety of scheduling problems, and there exist many extensions of it. A complete survey on different RCPSP variations and extensions is [HB10]. The authors identify six major kinds of variations of RCPSP from the literature, either from problems that have been defined with academic purposes or real life problems. The following is a small collection of all the possible variations collected in that work:

Generalised activity concepts The variations of this block change the definition of what is an activity. Some variations are: preemptive activities, i.e. their execution can be paused; variation of the resource demands over time; resource setup times before the execution of an activity; multiple available execution modes for an activity, where properties such as the duration or the resource demands of the activities are different in each execution mode; different forbidden execution periods for each activity.

Generalised temporal constraints There exist problems that do not only require end-start precedences but also other kinds temporal constraints over activities: minimal and maximal time lags, which require that the time difference between the starts of a source and a destination activities is not smaller (minimal) or greater (maximal) than a certain time lag; release dates and deadlines, that are minimum start time and maximum finish times of the activities that cannot be violated, or that can be violated at some penalty cost; time-switch constraints, enforcing that the activities can only start at particular instants of a cycle of work, for instance from Monday to Friday in a week.

Generalised resource constraints The RCPSP restricts to renewable resources that are occupied at a certain amount while an activity is running, and this amount is recovered once the activity finishes. We can find other variants: non-renewable resource constraints, such as a budget or raw material; cumulative resources which can both be consumed or refilled on some quantity by activities; resources with continuous capacity; resource capacities varying with the time.

Alternative objectives There exist many other optimisation criteria different than minimising the makespan: minimising the time difference from the desired start and end time of the activities w.r.t. the ones given by the schedule, or minimising the penalties for not respecting the desired bounds; optimising the robustness of solutions, in the sense that an unforeseen delay does not compromise the rest of the schedule; minimising or levelling the resource consumptions; weighted combinations of the different optimisation criteria.

Multiple projects There are some problem variations considering different projects at a time. Having multiple project adds some challenges such as combining different constraints or optimisation criteria, or executing only a subset of the projects while meeting the overall constraints.

In this thesis we tackle many problems extending the RCPSP which incorporate some of these variations:

- The Multi-mode RCPSP. This problem incorporates multiple execution modes (generalised activity concept) and non-renewable resource constraints (generalised resource constraints).
- The RCPSP with Time-Dependent Resource Capacities and Requests, Time-Dependent Resource Capacities and Requests. In this problem, the resource requests of activities vary during their execution (generalised activity concept) and the capacity of the resources vary during the project execution (generalised resource constraints).
- The Multi-mode RCPSP with Minimal and Maximal Time Lags. This problem generalises the Multi-mode RCPSP, by enabling time lags of arbitrary length between the starts of pairs of activities, either minimal or maximal (generalised temporal constraints).
- The Multi-Skill Project Scheduling Problem. In this problem, each resource is specialised in performing a set of skills, and the activities require a number of resources supplying each skill (generalised resource constraints).

Two different works [BDM+99, HDR99] provided two nomenclatures to identify the different possible variants of scheduling problems which support most of the previously mentioned variants. In particular, the RCPSP is denoted as $PS|prec|C_{max}$ in [BDM+99] and as $m, 1|cpm|C_{max}$ in [HDR99].

Benchmark Instances

Most of the benchmark datasets that are being used nowadays to evaluate the performance of solvers for RCPSP variants exists thanks to the work done in [KSD95]. There the authors introduced ProGen, an instance generator for the classical RCPSP as well as the multi-mode extension (MRCPSP). This tool was used to create PSPLIB [KS97], the most used library of scheduling problems. Many instances for other RCPSP variants have been created by extending the ones in PSPLIB, and other extensions of ProGen have been developed, such as ProGen/max [Sch98], which generates instances fo the variant with minimal and maximal time lags.

Solving Approaches

Back in the 1970s, the first exact formulations for scheduling problems were based on 0/1 Mixed Integer Linear Programming (MILP), i.e. MILP with variables than can only take value 0 or value 1. It was remarked by [Dav73] that although the formulation of RCPSP as a 0/1 MILP problem was common and had the advantage of being exact, solving it directly was impractical and unattractive due to its excessive computational requirement. For this reason the predominant technique was to use heuristic approximations to solve the problem. Nevertheless the 0/1 MILP approach was studied and many models were proposed. In many cases the MILP models were solved with specialised branch and bound procedures for RCPSP-like problems ([PH74, SDK78, TP78]). As stated in the survey [ISEZ93], by the late 1970s there were already more than 100 heuristic procedures and many exact solution techniques available for RCPSP.

More recently, the performance of exact approaches has been substantially improved, and these approaches are currently finding good results in challenging datasets. Moreover, even if the optimal solution is not found with a limited computation time, in many problems with size of practical interest the exact solvers are able to provide suboptimal solutions which can be competitive with the ones found using non-exact approaches. There is still a large number of recent works that tackle RCPSP and variants using MILP [BM08, KALM11, TSCS16]. However, alternative methods such as SAT/SMT or Constraint Programming (CP) are rapidly gaining acceptance

as a leading alternative to exact solving of CSPs and scheduling in particular. Many of the state-of-the-art last works on exact solving of RCPSP-based problems are based in SAT and SMT [ABP+11, CV11, Suy13], CP [VLS15], or Lazy Clause Generation (LCG) [SFS13, SFSW13]. In this thesis we compare the performance of our systems with other scheduling solvers that use the previously mentioned state-of-the-art exact solving methods. Therefore, now we provide an overview of the leading exact solving approaches in scheduling.

Constraint Programming

Constraint Programming (CP) is a programming paradigm where the relations between variables are stated in the form of constraints. Each variable can take values in a given domain, and each constraint further restricts combinations of values that a set of variables can take simultaneously. CP is devoted to solve *Constraint Satisfaction Problems* CSP, which are formally defined as a triple $\langle X, D, C \rangle$:

- $X = \{X_1, \ldots, X_n\}$ is a set of variables.
- $D = \{D_1, \ldots, D_n\}$ is a set of domains, where D_i is the domain of X_i.
- $C = \{C_1, \ldots, C_m, \}$ is a set of constraints.

A constraint $C_i \in C$ is a pair $\langle S_i, R_i \rangle$, where $S_i \subseteq X$ is the subset of variables involved in the constraint (*scope* of the constraint), and R_i is a relation over the variables in S_i. The *arity* of a constraint is the size of its scope, i.e., a constraint C_i with $|S_i| = k$ is a k-ary constraint. A relation R_i can be defined either extensionally as a set of allowed assignments to S_i, or intensionally as an expression that states the required relation between variables in S_i. An *assignment* of variables X is a mapping from variable X_i to a value in its domain D_i, for each variable X_i. A constraint C_i is *satisfied* if the assignment on variables in S_i satisfies relation R_i. A *solution* of a CSP is an assignment on X that satisfies all constraints in C. Most CP solving approaches implement a search procedure of the solution space which follows a backtracking scheme, in which there are integrated many techniques to enhance the performance. The most relevant are constraint propagation and consistency mechanisms, which aim at removing from the domains of the variables those values that do not have a support in some constraint, i.e., that do not belong to any assignment satisfying that constraint. This domain pruning is done as the search evolves.

A *global constraint* is a constraint that captures a relation between a non-fixed number of variables. For instance, the `allDifferent`$(X_1, \ldots, X_n)$ global constraint specifies that the values assigned to the variables $X_1, \ldots, X_n$ must

be pairwise distinct. Typically, a global constraint is semantically redundant in the sense that the same relation can be expressed as the conjunction of several simpler constraints[RVBW06]. Moreover, for some global constraints there exist efficient specialised propagators.

In the field of scheduling, and in particular in solving RCPSP-based problems, the global constraint *cumulative* [AB93] is widely used. It is defined as `cumulative(`S`,`D`,`R`,`B`)`, where:

- $S = \{S_1, \ldots, S_n\}$ is a set of integers denoting the start times of n activities.
- $D = \{D_1, \ldots, D_n\}$ is a set of integers durations of the activities.
- $R = \{R_1, \ldots, R_n\}$ is a set of integer resource requirements of the activities.
- B is the capacity of a renewable resource.

In the most common use of the constraint only S are variables and D, R, and B are input constants, but the global constraint accepts any of the parameters to be variables. The constraint is satisfied if the following holds for some range of time instants (scheduling horizon) H:

$$\sum_{\substack{i \in [1,n],\ s.t. \\ S_i \leq t < S_i + D_i}} R_i \leq B \qquad \forall t \in H \tag{2.1}$$

$$D_i \geq 0 \qquad \forall i \in [1, n] \tag{2.2}$$

$$R_i \geq 0 \qquad \forall i \in [1, n] \tag{2.3}$$

Among the most successful approaches to deal with the cumulative global constraint we find time-indexed decomposition [SFSW09], that we explain in detail un Subsection 2.1.3. Another approach that has given good results is *time-table edge finding* [Vil11], which is a wise combination of time-table and edge finding propagation techniques. The basic idea of time-table propagators is that, when a lower bound est_i and an upper bound lst_i on the start S_i are found, we know for sure that activity i is running (and consuming resources) during the time instants in the interval $[lst_i, est_i + D_i]$ if this interval is not null. By aggregating these intervals, it is possible to compute a minimum capacity profile (a timetable) which shows minimum resource usage over time, and that can detect infeasibility. On the other hand, edge finding propagators reason on sets of activities: given the capacity of a resource, and the duration and resource demands of a set of activities, it can be computed the minimum time interval required to execute the whole set of activities.

Lazy Clause Generation

Lazy Clause Generation (LCG) is a CP solving approach [FS09]. It also implements a backtracking scheme to explore the solution space and uses constraint propagators to prune the search space. LCG solvers also incorporate a SAT solver which is mainly used as a constraint propagator, but it can also perform other tasks such as driving the search with the variable activity based heuristic commonly used in SAT solvers (VSIDS). This can be done because finite domain variables are encoded as sets of Boolean variables: a variable X_i with domain $D_i = \{l, \ldots, u\}$ is represented using Boolean variables $[[X_i = l]], \ldots, [[X_i = u]]$, and $[[X_i \leq l]], \ldots, [[X_i \leq u - 1]]$. The variable $[[X_i = d]]$ is true if and only if X_i takes the value d. Similarly, the variable $[[X_i \leq d]]$ is true if and only if X_i takes a value less than or equal to d. Moreover, consistency of the semantics of these variables is enforced with expressions of the form $[[X_i \leq d]] \rightarrow [[X_i \leq d + 1]]$, and $[[X_i = d]] \leftrightarrow ([[X_i \leq d]] \wedge \neg[[X_i \leq d - 1]])$.

In recent years, LCG solvers such as G12 [SdlBM$^+$05] or Chuffed [Chu11] have been used to efficiently solve scheduling problems. In [SFS13] the authors presented a time-table edge finding propagator included in G12 which closed 6 instances of the RCPSP from PSPLib. In [SS16] and [YFS17], there were presented two CP models of the Multi-mode RCPSP and the Multi-Skill Project Scheduling Problem respectively which were state-of-the-art regarding performance, and also closed a number of open instances for such problems.

Failure-Directed Search

In [VLS15] there were presented CP solving approaches for many scheduling problems. They used the *IBM ILOG CPOptimizer* system which includes a search procedure specially designed for solving scheduling problems. The authors name this search procedure *Failure-Directed Search*(FDS). In FDS, the search does not operate on decision variables directly, but instead it works on a set of *binary choices*. The kind of choices used in FDS do not assign particular values to variables but split their domains (e.g. $S_2 \leq 4$). FDS tries to drive the search to conflicts in order to prove that the explored branch is infeasible resembling the *fail-first principle*. In this sense, it is also similar to the VSIDS heuristic of SAT solvers, which also prioritises the exploration of assignments which most likely will lead to a conflict. The search algorithm maintains a rating for each choice, which is smaller for choices that quickly lead to a conflict, and picks the choices with smallest ratios.

Using this system, the authors solved many scheduling problems, including the RCPSP and its extensions Multi-mode RCPSP, RCPSP/max and Multi-

Mode RCPSP/max. They were able to improve the best known bounds for some instances of these problems, being especially remarkable the case of the Multi-mode RCPSP/max, where they were able to close all 85 open instances. For this last problem, the authors detected that the existing instances were not hard regarding resource constraints, and obtained very good results by solving a MILP relaxation of the problem as a preprocessing step.

2.1.2 RCPSP

The Resource-Constrained Project Scheduling Problem (RCPSP) can be formally defined by a tuple (V, p, E, R, B, b) where:

- $V = \{A_0, A_1, \ldots, A_n, A_{n+1}\}$ is a set of activities. Activities A_0 and A_{n+1} are dummy activities introduced by convention, which represent the start and the end of the schedule respectively. They don't consume resources and have duration 0. The set of non-dummy activities is defined by $A = \{A_1, \ldots, A_n\}$.
- $p \in \mathbb{N}^{n+2}$ is a vector of naturals, where p_i is the duration of A_i.
- E is a set of pairs of activities representing end-start precedence relations. Concretely, $(A_i, A_j) \in E$ iff the execution of activity A_i must precede that of activity A_j, i.e., activity A_j must start after activity A_i has finished. We assume that we are given an activity-on-node *precedence graph* $G = (V, E)$ that contains no cycles, since otherwise the precedence relation is inconsistent. By convention there is a path from A_0 to any other activity, and also a path from any activity to A_{n+1}.
- $R = \{R_1, \ldots, R_v\}$ is a set of renewable resources.
- $B \in \mathbb{N}^v$ is a vector of naturals, where B_k is the available amount of each resource R_k.
- $b \in \mathbb{N}^{(n+2)\times v}$ is a matrix of naturals corresponding to the resource demands of activities, where $b_{i,k}$ represents the amount of resource R_k that activity A_i is using per time step during its execution.

It is common in the literature to compute the transitive closure E^* of E, which contains a tuple $(A_i, A_j, l_{i,j})$ iff there is a path from A_i to A_j in G. The *time lag* $l_{i,j}$ is the critical path from A_i to A_j in G. The critical path from A_i to A_j is the path of maximum sum of weights, where the weight of an edge (precedence) is the duration of the predecessor activity. We will denote

$G^* = (V, E^*)$ as the *extended precedence graph*, which has a weight $l_{i,j}$ for each edge $(A_i, A_j) \in E^*$.

A schedule is a vector of naturals $S = (S_0, S_1, \ldots, S_n, S_{n+1})$ where S_i denotes the start time of activity A_i. We assume that $S_0 = 0$. A solution of the RCPSP is a schedule S of minimal makespan S_{n+1} subject to the precedence and resource constraints. More precisely, the constraints can be formally stated as:

Minimise: $$S_{n+1} \tag{2.4}$$

Subject to:

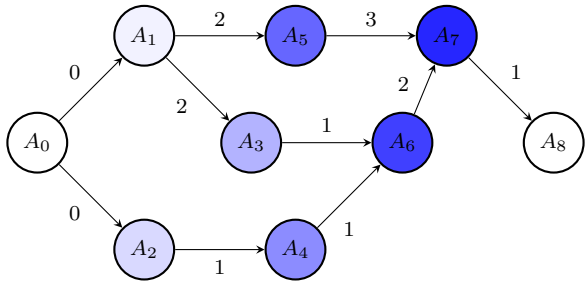
$$S_j - S_i \geq p_i \qquad \forall (A_i, A_j) \in E \tag{2.5}$$

$$\sum_{A_i \in A} ite\big(S_i \leq t < S_i + p_i;\, b_{i,k};\, 0\big) \leq B_k \qquad \forall R_k \in R, \forall t \in H \tag{2.6}$$

where $ite(c;\, e_1;\, e_2)$ is an *if-then-else* expression denoting e_1 if c is true and e_2 otherwise, $H = \{0, \ldots, UB\}$ is the scheduling horizon, i.e. a wide enough time period to schedule the project. Precedence constraints (2.5) state that, for any pair $(A_i, A_j) \in E$, activity A_j cannot start until A_i has finished. The renewable resource constraints (2.6) state that the capacities of the renewable resources cannot be exceeded at any time.

Figure 2.1 illustrates an example RCPSP instance with 7 non-dummy activities and 2 resources, and Figure 2.2 shows an optimal solution for this instance.

The RCPSP is an NP-hard problem in the strong sense [GJ75, BLK83b], although just finding a solution that satisfies the precedence and resource constraints can be done in polynomial time if there is no constraint over the makespan.



	A_0	A_1	A_2	A_3	A_4	A_5	A_6	A_7	A_8
p_i	0	2	1	1	1	3	2	1	0
$b_{i,1}$	0	3	3	2	1	2	0	3	0
$b_{i,2}$	0	2	2	4	3	1	2	2	0

B_1	3
B_2	4

Figure 2.1: Example RCPSP instance with 7 non-dummy activities and 2 resources.

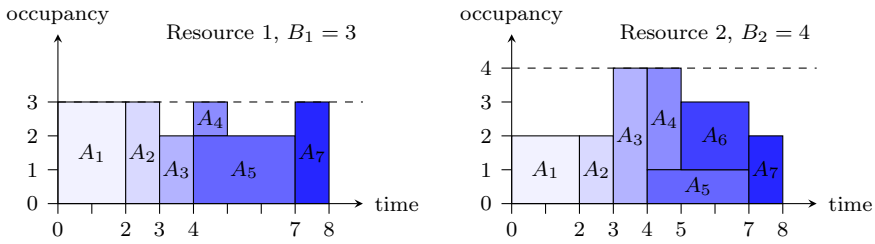


Figure 2.2: Optimal solution of the instance of Figure 2.1, with schedule $S = (0, 0, 2, 3, 4, 4, 5, 7, 8)$. The capacity of the resources is never exceeded, and the precedence relations are respected.

2.1.3 Modelling Renewable Resource Constraints

The constraint over renewable resources in the RCPSP is the most challenging part of this problem, and it has been widely studied. In [Suy13] we can find examples of the principal modelling viewpoints that have been used: the *Task* approach consists in checking the consumption of the resources at the start time of every activity; a similar approach is *Event*, which consists in representing the schedule as a finite number of events (e.g. time instants where the activities can start), then schedule the events, distribute the activities among events, and finally ensure that the resource constraints are satisfied at all events; the *Flow* viewpoint models a network flow that, once an activity finishes, reassigns the resources that it is using to some other activities.

In this thesis we have focused in time-indexed models, usually known as the *Time* approach [PW96], which have shown to provide good performance when solving the different benchmark instances available in the literature [ABP+11, SFSW09]. This approach consists in discretising the *scheduling horizon* H —the period of time in which the schedule will take place— into unit intervals from 0 to an *upper bound* UB, i.e. $H = \{0, 1, \ldots, UB\}$. The upper bound must be large enough so that the optimal solution (if any) will have a makespan not greater than UB. Then, it must be ensured that the capacity of a resource is not exceeded at any of the time instants in H. A usual way of achieving this purpose is to introduce an auxiliary 0/1 variable for each activity and for each time instant. There are different semantics that can be reified to the auxiliary time-indexed variables, leading to different constraints [Art13]. In the formulations introduced we will be mainly using variables with (extensions of) the following semantics:

$x_{i,t}$**:** 0/1 variable which is true iff activity A_i is running at time t.

Then, the constraint over a resource R_k can be expressed as:

$$\sum_{A_i \in V} b_{i,k} \cdot x_{i,t} \leq B_k \qquad \forall t \in H \tag{2.7}$$

Constraints (2.7) ensure that the added demand of activities running at a particular time t is not greater than the capacity of a particular resource R_k.

Time Windows

One of the key points of the Time approach is only introducing variables for the time instants at which an activity can be running. This will both reduce the number of auxiliary variables and the size of the constraints. For this

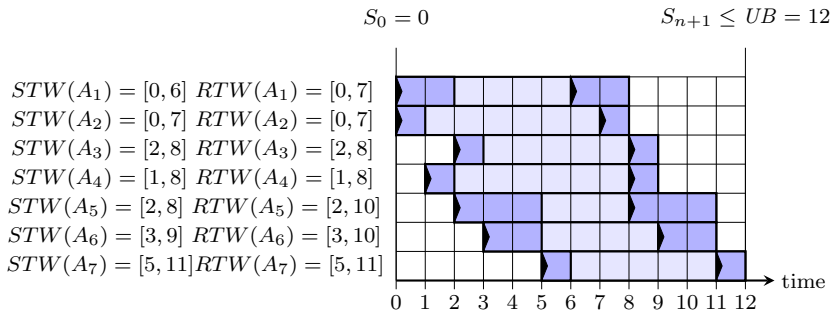


Figure 2.3: Illustration of the start and running time windows of the non-dummy activities of Figure 2.1, with an upper bound of 12. The coloured cells are the time instants at which the activity can be running (RTW), being the two dark coloured rectangles the earliest and latest possible schedule of each activity. STW is all the starts comprised between the two black arrows, which denote earliest and latest start time.

purpose it is usual to pre-compute the time instants at which an activity can be running (*run time window*) or can start (*start time window*). Given the extended precedence graph G^* and an upper bound for the makespan UB, we can define the earliest/latest start/close time of each activity, and its time windows as:

$ES(A_i)$ *Earliest start time* of activity A_i, equal to $l_{0,i}$.

$LS(A_i)$ *Latest start time* of activity A_i, equal to $UB - l_{i,n+1}$.

$EC(A_i)$ *Earliest close time* of activity A_i, equal to $l_{0,i} + p_i$.

$LC(A_i)$ *Latest close time* of activity A_i, equal to $UB - l_{i,n+1} + p_i$.

$RTW(A_i)$ *Run time window*: set of time instants at which an activity can be running, that is $ES(A_i) \dots LC(A_i) - 1$.

$STW(A_i)$ *Start time window*: set of time instants at which an activity can start, that is $ES(A_i) \dots LS(A_i)$.

Figure 2.3 illustrates the time windows of the RCPSP instance of Figure 2.1.

Computation of Upper Bound

Another key point in the Time approach is being able to identify a good upper bound for the makespan. Note that the latest start times are proportional to the upper bound, and in consequence also the size of the time windows

increase with a large UB. Moreover the number of constraints like (2.7) also depend on the scheduling horizon.

In the RCPSP there is no constraint on how late an activity can start if the schedule is not limited by a maximum makespan. Therefore a trivial upper bound can be obtained from a schedule in which only one activity runs at a time and there are no periods of inactivity. If the problem instance is not inconsistent, in the sense that there are no cycles in the precedence graph and there is no activity requiring more units of a resource than its availability, then a schedule can be easily constructed with a breadth first traversal of the precedence graph. In any such schedule, the makespan is the sum of the durations:

$$\sum_{A_i \in A} p_i \tag{2.8}$$

This is a valid value for UB, but it will generally be very far from the optimal makespan. To overcome this issue, in this thesis we will be using a fast greedy heuristic to get a first schedule from which we can infer a first UB, which will be much smaller than the trivial upper bound (2.8). This heuristic is the *Parallel Scheduling Generation Scheme* (PSGS) proposed in [Kel63] and described in [Kol96]. Given a project of n activities, this method requires at most n stages to find a schedule, and at each stage a subset of the activities are scheduled. Each stage s has associated a schedule time t_s (where $t_{s'} \leq t_s$, for $s' \leq s$). There are three activity sets that are updated as the algorithm runs:

- Complete set C: activities already scheduled and completed up to the schedule time t_s.
- Active set A: activities already scheduled, but still active at the schedule time t_s.
- Decision set D: activities not yet scheduled which are available for scheduling with start time t_s, w.r.t. precedence and resource constraints.

Each stage consists of two steps:

1. Defining the new t_s as the earliest completion time of activities in the active set A —the first stage starts at $t_0 = 0$. The activities with a finish time equal to the new t_s are removed from A and put into C. This may place new activities into D.

2. One activity from D is selected by some priority rule —we will be using input order—, and scheduled to start at t_s, being removed from D and added to A. Then set D is recomputed. This step is repeated until D becomes empty.

The method terminates when all activities are scheduled.

2.2 Boolean Satisfiability (Modulo Theories)

In this section we give an overview on basic concepts on Boolean Satisfiability (SAT) and its extension Satisfiability Modulo Theories (SMT). The reader can refer to [BHvMW09] for deeper concepts covering this area. First, in Section 2.2.1 we present SAT with all the notation and definitions that we will be using hereinafter, as well as the basic algorithms involved in SAT solving. Then we introduce its extension to SMT in Section 2.2.2.

2.2.1 SAT

A *Boolean variable* is a variable than can take truth values 0 (*false*) and 1 (*true*). A *literal* is a Boolean variable x or its negation $\overline{x}$. A *clause* is a disjunction of literals $l_1 \vee \cdots \vee l_n$, which can also be presented in the equivalent form $\overline{l_1} \wedge \cdots \wedge \overline{l_m} \rightarrow l_{m+1} \vee \cdots \vee l_n$. A *propositional formula in conjunctive normal form* (CNF), in short a *Boolean formula* hereinafter, is a conjunction of clauses $c_1 \wedge \cdots \wedge c_n$. Clauses are usually seen as sets of literals, and formulas as sets of clauses. A Boolean formula represents a *Boolean function* $f : \{0,1\}^n \rightarrow \{0,1\}$.

An *assignment* or *interpretation* of a formula is a mapping of Boolean variables to truth values; it can also be seen as a set of literals (e.g., $\{x = 1, y = 0, z = 0\}$ is usually denoted $\{x, \overline{y}, \overline{z}\}$). A *total assignment* has a truth value mapped to all variables of the formula, while a *partial assignment* maps a proper subset of the variables. A *satisfying assignment* or *model* of a formula is an assignment that makes it evaluate to 1. In particular, an assignment A satisfies a formula F in CNF if at least one literal of each clause in F belongs to A.

Definition 2.2.1. *The problem of Boolean Satisfiability (SAT) is the problem of determining if there exists a satisfying assignment for a given Boolean formula.*

Given two Boolean formulas F and G, we say that G is a logical consequence of F, written $F \models G$, iff every model of F is also a model of G. We say

that two Boolean formulas F and G are logically equivalent, denoted $F \equiv G$, if $F \models G$ and $G \models F$. An assignment can also be seen as a conjunction of literals and therefore a Boolean Formula. Hence, if an assignment A is a model of a formula F, it is denoted as $A \models F$.

We say that a formula G is an *encoding of a Boolean function* F if the following holds: given an assignment A over the variables of F, A satisfies F iff A can be extended to a satisfying assignment of G.

In this thesis we often deal with *constraints on Boolean variables*. A constraint C on a set of Boolean variables X can be defined extensionally as a set of assignments on X that satisfy C, or intentionally as a Boolean formula on X. Therefore a constraint on Boolean variables represent a Boolean function, which evaluates to *true* if and only if the constraint is satisfied with the given assignment. Therefore, we define the *encoding of a constraint on Boolean variables* as the encoding of the Boolean function represented by such constraint.

The Resolution Rule

The *Resolution* is an inference rule which, given two clauses c_1, c_2 containing complementary literals, produces a new clause which is a logical consequence of $c_1 \wedge c_2$. Two literals are said to be complements if one is the negation of the other (in the following $\overline{p}$ is taken to be the complement of p).

Example 1. *Consider the two clauses* $p \vee q_1 \vee \cdots \vee q_n$ *and* $\overline{p} \vee r_1 \vee \cdots \vee r_n$. *As they have complementary literals, we can apply the resolution rule:*

$$\frac{p \vee q_1 \vee \cdots \vee q_n \quad \overline{p} \vee r_1 \vee \cdots \vee r_n}{q_1 \vee \cdots \vee q_n \vee r_1 \vee \cdots \vee r_n}$$

and produce the clause $q_1 \vee \cdots \vee q_n \vee r_1 \vee \cdots \vee r_n$.

The produced clause is called a *resolvent*, and the dividing line stands for logical entailment.

When coupled with a complete search algorithm [DP60], the resolution rule yields a sound and complete algorithm for deciding the satisfiability of a propositional formula in CNF. Although modern SAT solvers are not based in this approach the resolution rule still plays an important role, as we will see in the when Conflict Driven Clause Learning is explained.

Model Search Based Algorithms

The basic procedure behind the algorithms for SAT solving which are used currently is a backtracking scheme. The search for the solution consists in trying to construct an assignment which satisfies all the clauses. Therefore there

is a partial assignment which is modified as the search evolves, and the algorithm finishes when either a model is found, i.e. the formula is satisfiable, or it has been checked that no assignments is a model, i.e. the formula is unsatisfiable. A key point for the efficiency of this approach is avoiding enumerating all possible assignments by applying different deduction techniques.

Unit Propagation

Unit propagation (UP) is the core deduction mechanism in modern SAT solvers. It can be applied whenever given an assignment a clause has all the literals but one assigned with *false*, and the remaining one is unassigned. In this case, UP sets the remaining literal to *true* as it is the only way of satisfying the clause.

Example 2. *Consider the clause $p \vee q \vee \overline{r}$ and the assignment $\{\overline{p}, r\}$. UP will extend the assignment as $\{\overline{p}, r\} \cup \{q\}$.*

When constructing encodings of constraints on Boolean variables it is crucial that the resulting formula lets UP make strong deductions such as maintaining *Generalised Arc Consistency* (GAC).

Definition 2.2.2. *An encoding E of a constraint C on Boolean variables is said to* UP-maintain GAC *if it satisfies the following property: given a partial assignment A, if a variable x of C is true (respectively false) in every extension of A satisfying C, then unit propagating A on E will extend A to $A \cup \{x\}$ (respectively $A \cup \{\overline{x}\}$) [BBR09].*

The Davis-Putnam-Logemann-Loveland Algorithm

Since SAT is an NP-complete problem [Coo71], only algorithms with exponential worst-case complexity are known to solve it. The *Davis-Putnam-Logemann-Loveland* (DPLL) procedure [DLL62] is a procedure to decide if a CNF is satisfiable, and if it is, it provides satisfying assignment. This algorithm implements a backtracking scheme which starts with an empty assignment. It extends the assignment with UP until a fix point is reached, and then it branches (does a *decision*) on an unassigned literal (*branching literal*) and recursively repeats the process. The number of decisions that have been taken at a point of the search receives the name of *decision level*. If a clause becomes unsatisfied during the search (*conflict*), the algorithm backtracks to the last branching and tries the search negating the branching literal. If there is no backtrack point before a conflict, i.e. if the search decision level is 0,

Algorithm 1 DPLL

Input: F : CNF formula, A : partial assignment
Output: unsatisfiable or a model of F

```
1: A ← Unit-Propagation(F,A)
2: if hasUnsatisfiedClause(F,A) then
3:    return unsatisfiable
4: else if allClausesSatisfied(F,A) then
5:    return A
6: end if
7: l ← PickBranchingLiteral(F, A)
8: A' ← DPLL(F, A ∪ {l})
9: if A' is a model then
10:   return A'
11: else
12:   A' ← DPLL(F, A ∪ {l̄})
13:   if A' is a model then
14:      return A'
15:   else
16:      return unsatisfiable
17:   end if
18: end if
```

it is proved that the formula is unsatisfiable. Algorithm 1 shows the DPLL procedure, whose first call should receive an empty assignment as input.

The weak point of DPLL is that it does not memorise if a substructure of the search tree have lead to a conflict, and the same dead-ended set of choices can be explored multiple times. The Conflict-Driven Clause-Learning algorithm incorporates learning techniques to overcome this weakness.

The Conflict-Driven Clause-Learning Algorithm

The main improvement of *Conflict-Driven Clause-Learning* (CDCL) algorithm with respect to DPLL is that it keeps track of the reason why each one of the literals in the partial assignment have been assigned, that can be either the result of a decision, or the application of UP on a particular clause. When the algorithm reaches a conflict, this information can be used to perform a *conflict analysis*: a chain of resolution steps is applied starting from the conflicting clause to derive a *lemma*, that is a clause which explains the reason of the conflict. The lemma is added to the formula (*clause learning*) to prevent repeating the same conflicting assignments. Then a *non-chronological backtracking*

Algorithm 2 CDCL

Input: F : CNF formula
Output: unsatisfiable or a model of F

```
1:  A ← {}
2:  while not AllVariablesAssigned(F,A) do
3:    A ← Unit-Propagation(F,A)
4:    if HasUnsatisfiedClause(F,A) then
5:      decision_level ←ConflictAnalysisAndLearning(F,A)
6:      if decision_level < 0 then
7:        return unsatisfiable
8:      else
9:        NonChronologicalBacktrack(F,A,decision_level)
10:     end if
11:   else if not AllVariablesAssigned(F,A) then
12:     l ← PickBranchingLiteral(F, A)
13:     A ← A ∪ {l}
14:   end if
15: end while
16: return A
```

is performed , i.e. it does not only undo the last decision but continues the backtrack until the derived lemma is no longer falsified. Therefore, the search can step back many decision levels after a conflict.

Non-chronological backtracking was originally proposed as a technique for solving CSPs [SS77], and it was successfully incorporated in SAT solvers [MSS99]. The conflict analysis implementation which has shown to be most successful, and that modern SAT solvers implement, is the one of finding the *first unique implication point* (1UIP) [MMZ$^+$01]. Algorithm 2 shows the structure of the typical CDCL algorithm.

Many other features have been added to CDCL to improve the performance of SAT solvers. Among the most important we can identify the use of the efficient branching heuristic VSIDS and the inclusion *restart* policies [MMZ$^+$01]. The *Variable State Independent Decaying Sum* (VSIDS) is a heuristic to pick the branching variables in CDCL. Each literal has an activity value associated, which is increased when the literal is involved in a conflict, and the solver branches on the most active variables. Moreover, all the activity values are periodically divided by a constant. This way, the search is guided to decide on literals which have been involved in recent conflicts. Regarding restarts, they consist in forgetting all the current assignments and then start the search again at the root of the search tree. After a restart some information is pre-

served, most notably the previously learned conflict-driven clauses and the activity values. Intuitively, restarting prevents the solver from being stuck in an area of the search space that contains no solution.

2.2.2 Satisfiability Modulo Theories

Satisfiability Modulo Theories (SMT) is an extension of SAT, where the satisfiability of a first order logic formula has to be determined. This formula can contain not only Boolean variables, but also Boolean predicates with predefined interpretations from background theories, or *theory atoms*. The following is an example formula which contains atoms of the theory of Linear Integer Arithmetic (LIA):

$$(x \geq 3 \vee \overline{q}) \wedge (\overline{y = 4} \vee p)$$

where x and y are integers, while p and q are Boolean variables.

Definition 2.2.3. *A* theory *is a set of first-order formulas closed under logical consequence. A theory T is said to be* decidable *if there is an effective method for determining whether arbitrary formulas are included in T.*

Definition 2.2.4. *A formula F is* T-satisfiable or T-consistent *if $T \cup \{F\}$ is satisfiable in the first-order sense. Otherwise, it is called* T-unsatisfiable or T-inconsistent*.*

Definition 2.2.5. *If A is a T-consistent partial truth assignment and F is a formula such that $A \models F$, i.e., A is a (propositional) model of F, then we say that A is a* T-model *of F.*

Definition 2.2.6. *The* SMT problem for a theory T *is the problem of determining, given a formula F, whether F is T-satisfiable.*

Theories

The *Satisfiability Modulo Theories Library* (SMT-LIB) [BST10] has the goal of establishing a common standard for the specification of benchmarks and of background theories, as well as to establish a library of benchmarks for SMT. Currently the widely accepted standard SMT language is SMT-LIB2. Some of the theories supported by SMT-LIB2 are:

- The theory of *Equality and Uninterpreted Functions* (QF_EUF, or simply QF_UF). It is the quantifier-free fragment of first order logic with equality and no restrictions on the signature (hence the name UF for Uninterpreted Functions). Uninterpreted functions have no other property than its name and arity, and are only subject to the following axiom: $x_1 = x'_1 \wedge \cdots \wedge x_n = x'_n \rightarrow f(x_1, \ldots, x_n) = f(x'_1, \ldots, x'_n)$.

- *Linear Arithmetic* over the integers (QF_LIA) or the reals (QF_LRA). It comprises quantifier-free formulas with Boolean combinations of inequations between linear polynomials of the form $\sum_{i=1}^{n} a_i x_i \# c$, where a_i and c are integer (or real) constants, x_i integer (or real) variables and $\#$ any relational operator of $\{<, >, \leq, \geq, =\}$. An example formula in QF_LIA is $(3x + 4y \geq 7) \rightarrow (z = 3)$, where x, y and z are integer variables. Many SMT solvers use specialised Simplex like algorithms [DdM06b] to deal with satisfiability checks of conjunctions of LIA atoms (for instance Yices 2 [Dut14]).

- *Difference Logic* over the integers (QF_IDL) or the reals (QF_RDL). It is an efficiently solvable fragment of linear arithmetic. Here atoms are restricted to have the form $x - y \# k$, where x and y are numeric (integer or real) variables, k is a numeric (integer or real) constant and $\# \in \{=, <, >, \leq, \geq\}$.

The Lazy SMT Approach

There are two principal types of procedures for solving SMT, the so-called *eager* and *lazy* approaches. In the eager approach, the input formula is fully translated into a propositional CNF formula, whose satisfiability is then checked by a SAT solver. Sophisticated ad-hoc translations have been developed for several theories, but still on many practical problems either the translation process or the SAT solver run out of time or memory [dMR04]. On the contrary, the lazy approach consists in integrating a T-solver, i.e. a decision procedure for the given theory T, in a SAT solver. Currently most successful SMT solvers are essentially based on a lazy approach. In this approach, while the SAT solver is in charge of the Boolean component of reasoning, the T-solver deals with conjunctions of literals that belong to T. It is named *lazy* because the theory information is only used when checking the consistency of the truth assignment against the theory T. The basic idea is to let the T-solver analyse the partial truth assignment that the SAT solver is building, and warn about conflicts with the theory T (T-inconsistency). This idea combines the

Algorithm 3 Lazy Bool+T Algorithm

Input: F : SMT formula
Output: Satisfiability of F

1: $\alpha^p \leftarrow T2B(Atoms(F))$;
2: $F^p \leftarrow T2B(F)$;
3: **while** *Bool-satisfiable*(F^p) **do**
4: $A^p \leftarrow pick\_total\_assignment(\alpha^p, F^p)$;
5: $A \leftarrow B2T(A^p)$;
6: $(\rho, \pi) \leftarrow T$-*satisfiable*(A);
7: **if** $\rho =$ `sat` **then**
8: **return** `sat`;
9: **else**
10: $F^p \leftarrow F^p \wedge \overline{T2B(\pi)}$;
11: **end if**;
12: **end while**
13: **return** `unsat`;

efficiency of the SAT solver and special-purpose algorithms inside the T-solver for non-Boolean reasoning.

Algorithm 3 shows a simplified enumeration-based T-satisfiability procedure (from [BCF$^+$06]), where the T-consistency is only checked for total Boolean assignments. The reader is referred to [Seb07] for a survey on the lazy SMT approach. The algorithm enumerates the Boolean models of the propositional abstraction of the SMT formula F and checks for their satisfiability in the theory T.

- The function *Atoms* takes a quantifier-free SMT formula F and returns the set of atoms which occur in F, where an atom is either a propositional variable or an expression of theory T.
- The function $T2B$ maps propositional variables to themselves, and ground atoms into fresh propositional variables. It is homomorphic with respect to Boolean operators and set inclusion.
- F^p is initialised to be the propositional abstraction of F using $T2B$.
- The function $B2T$ is the inverse of $T2B$.
- A^p denotes a propositional assignment as a set (conjunction) of propositional literals.

- The function *pick_total_assignment* returns a total assignment to the propositional variables in F^p. In particular, it assigns a truth value to all variables in α^p.

- The function *T-satisfiable* checks if a set of conjuncts A is *T-satisfiable*, i.e., if there is a model for $T \cup A$, returning (`sat`,$\emptyset$) in the positive case and (`unsat`,π) otherwise, being $\pi \subseteq A$ a *T-unsatisfiable* set (the theory conflict set). Note that the negation of the propositional abstraction of π is added to F^p in case of `unsat` (learning).

In the approach presented so far, the T-solver provides information only after a T-inconsistent total assignment has been generated. In this sense, the T-solver is used only to validate the search a posteriori, not to guide it a priori. In order to overcome this limitation, the T-solver could also be used to detect literals l occurring in F such that $M \models_T l$, where M is a partial assignment of F. This is called *theory propagation*. The propagation capability is a very important aspect of theory solvers, since getting more general explanations (conflict sets) from the theory solver is essential in order to keep the learned lemmas as short as possible and will allow for more pruning in general. In practice, the enumeration of Boolean models is carried out by means of efficient implementations of the CDCL algorithm [ZM02], where partial assignments A^p are incrementally built. On the other hand, the T-solver checks the consistency of the assigned literals of the theory. These systems benefit of the spectacular progress in performance from SAT solvers in the last decade, achieved thanks to better implementation techniques and conceptual enhancements.

2.3 Pseudo-Boolean Constraints and Decision Diagrams

In this section we first present basic notions and notation about pseudo-Boolean constraints. Since decision diagram based SAT encodings of pseudo-Boolean constraints are a core subject of this thesis, we also devote a section on preliminaries of Binary Decision Diagrams, and another section to review the state-of-the-art on decision diagram based SAT encodings.

2.3.1 Pseudo-Boolean Constraints

Definition 2.3.1. *A* pseudo-Boolean *(PB) constraint is a Boolean function of the form* $\sum_{i=1}^{n} q_i x_i \# K$*, where* $\# \in \{<, \leq, =, \geq, >\}$*,* $q_1, \ldots, q_n$ *and* K *are integer constants, and* $x_1, \ldots, x_n$ *are 0/1 variables.*

By the *scope* of a constraint, we mean the set of variables appearing in the constraint.

Checking the satisfiability of PB constraints with $=$ operator is harder than doing so with constraints with other operators (unless P=NP), since the *subset sum* problem, which is NP-complete, is polynomially reducible to the former constraints. In fact, an equality PB constraint is equivalent to the conjunction of two inequality PB constraints, i.e. $(\sum_{i=1}^{n} q_i x_i = K) \equiv (\sum_{i=1}^{n} q_i x_i \geq K \wedge \sum_{i=1}^{n} q_i x_i \leq K)$. By contrast, deciding the satisfiability of PB constraints with operators $<, \leq, \geq, >$ is trivial. For instance, a constraint of the form $\sum_{i=1}^{n} q_i x_i \leq K$ will be satisfiable if and only if the assignment $\{x_i \mid i \in 1..n, q_i < 0\} \cup \{\overline{x_i} \mid i \in 1..n, q_i \geq 0\}$ is a model of it. In this thesis we mainly deal with PB constraints expressing inequalities. Therefore, unless otherwise stated, the different definitions and statements over PB constraints hereinafter refer to inequality PB constraints, and might not apply to equality PB constraints.

A particular case of PB are *cardinality constraints*, whose satisfiability is trivially decidable also for $=$ operator.

Definition 2.3.2. *A* cardinality constraint *is a Boolean function of the form* $\sum_{i=1}^{n} x_i \# K$*, where* $\# \in \{<, \leq, =, \geq, >\}$*,* K *is an integer constant, and* $x_1, \ldots, x_n$ *are 0/1 variables.*

Among cardinality constraints we define *at-most-one* (AMO), *at-least-one* (ALO), and *exactly-one* (EO) constraints.

Definition 2.3.3. *An* at-most-one *(AMO) constraint is a Boolean function of the form* $\sum_{i=1}^{n} x_i \leq 1$*, where all* x_i *are* $0/1$ *variables.*

Definition 2.3.4. *An* at-least-one *(ALO) constraint is a Boolean function of the form* $\sum_{i=1}^{n} x_i \geq 1$*, where all* x_i *are* $0/1$ *variables.*

Definition 2.3.5. *An* exactly-one *(EO) constraint is a Boolean function of the form* $\sum_{i=1}^{n} x_i = 1$*, where all* x_i *are* $0/1$ *variables. It can be defined as the conjunction of an AMO constraint and an ALO constraint.*

PB constraints can always be normalised to the form $\sum_{i=1}^{n} q_i x_i \leq K$, where $q_i \geq 0$ [ES06]. Most existing SAT encodings of PB constraints require the input PB to be in this normal form. The normalisation procedure is the following:

- Strict inequalities are converted to non-strict by adding/subtracting 1 to K.

- $\sum_{i=1}^{n} q_i x_i \geq K$ is converted to $\sum_{i=1}^{n} -q_i x_i \leq -K$.
- Negative coefficients are removed by substituting x_i for $1 - \overline{x_i}$, and modifying K accordingly.

Note that after the normalisation a PB may not only contain variables but also negations of variables, as literals. For simplicity we assume w.l.o.g. that PB constraints contain variables, since a literal also can take values 0/1 (false/true), and a literal $\overline{x}$ could be rewritten as a variable $x' \leftrightarrow \overline{x}$.

A normalised PB constraint is a tautology if $\sum_{i=1}^{n} q_i \leq K$, and it is unsatisfiable if $K < 0$. Also, it is a *monotonic decreasing function*, meaning that given any assignment which satisfies the constraint, it will still be a model after changing to *false* the value of any variable assigned with *true*.

PB constraints appear frequently in formulations of CSPs, especially when a bound is required on the added cost of a set of Boolean choices. For instance, in *knapsack* problems [HLS10] a choice has to be made about whether an item is packed, in which case it will occupy some of the limited space of the knapsack. Another example is *routing* problems [Lap92], where deciding to go from one location to another increases in a particular amount the total travelled distance. As seen in Section 2.1, PB constraints appear in scheduling problems when formulating resource constraints.

2.3.2 Binary Decision Diagrams for Pseudo-Boolean Constraints

A PB constraint expressed in the form $\sum_{i=1}^{n} q_i x_i \# K$ naturally fits in some modelling languages such as Mixed Integer Linear Programming (MILP) or Constraint Programming (CP). However, if they are to be included in a SAT formulation, a CNF encoding of the constraint needs to be provided.

A well-known approach to translate PB constraints to CNF is by means of Binary Decision Diagrams (BDD) [Bry86, ES06, ANO+12]. A BDD is a data structure that can represent the evaluation of all the possible assignments of any given Boolean function, in particular of a PB constraint. Then, the encoding is obtained by generating a set of clauses which model the semantics of the diagram.

Definition 2.3.6. *A* Binary Decision Diagram *(BDD) is a rooted, directed, acyclic graph which represents a Boolean function. BDDs have two terminal nodes, namely* $\mathcal{F}$-terminal *and* $\mathcal{T}$-terminal*. Each non-terminal node has an associated variable (*selector*), and has two outgoing edges, representing the true and the false assignment of the selector. Every truth assignment of the*

variables follows a path from the root to the $\mathcal{T}$-terminal when it satisfies the formula, or to the $\mathcal{F}$-terminal otherwise.

A BDD is *ordered* if different variables appear in the same order on all paths from the root. A BDD is said to be *reduced* if it satisfies the following two conditions:

- It contains no isomorphic sub-BDDs.
- There is no node whose true and false children are the same.

A *Reduced Ordered Binary Decision Diagram* (ROBDD) is canonical (i.e. unique) for a particular Boolean function and variable order [ANO+12].

Figure 2.4 contains a non-reduced ordered BDD representation of the PB constraint $C : 2x_1+3x_2+4x_3+6x_4 \leq 7$ with the variable order $x_1 \prec x_2 \prec x_3 \prec x_4$. Figure 2.5 shows the ROBDD representation of the same PB constraint and variable order. As seen in the pictures, an ordered BDD can be organised in different *layers*, where at each layer a different selector is considered. For instance, in all the nodes of the second layer we choose whether to set x_2 to 1 or to 0.

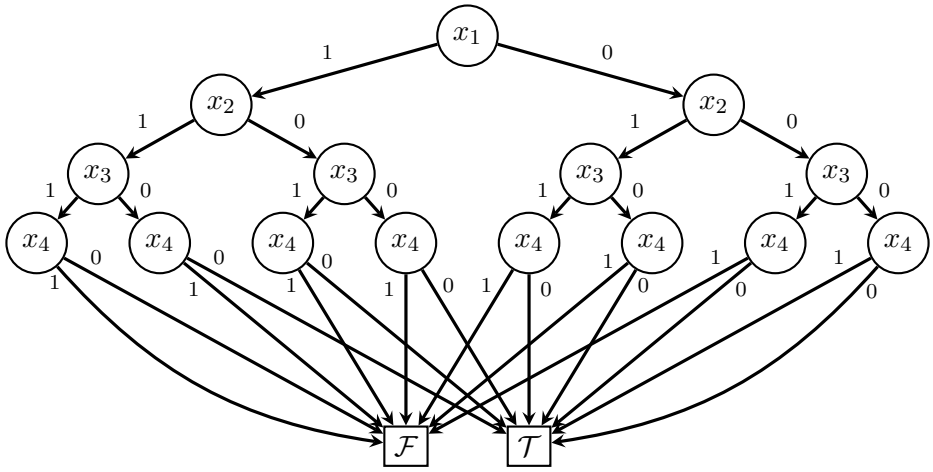


Figure 2.4: Non-reduced Ordered BDD for the PB constraint $2x_1+3x_2+4x_3+6x_4 \leq 7$ with order $x_1 \prec x_2 \prec x_3 \prec x_4$.

No polynomial time algorithm is known to construct ROBDDs representing Boolean functions in general, but when the diagram represents a PB constraint it can be constructed in polynomial time w.r.t. the size of the final ROBDD by means of dynamic programming [ANO+12]. There exist PB constraints

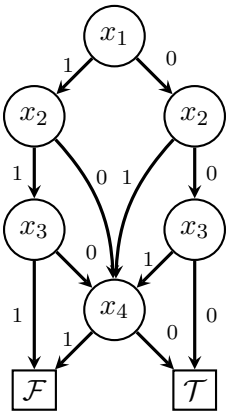


Figure 2.5: ROBDD for the PB constraint $2x_1 + 3x_2 + 4x_3 + 6x_4 \leq 7$ with order $x_1 \prec x_2 \prec x_3 \prec x_4$.

which only have ROBDD representations of exponential size in the number of variables [HTY94], and hence the construction time can be exponential in the number of variables of the PB in the worst case. However this is not an issue in many practical applications, because the number of nodes of ROBDD representing a PB constraint of the form $\sum_{i=1}^{n} q_i x_i \leq K$ is also $O(nK)$. This means that, in order to get a ROBDD with exponential size, it is required on the one hand that K is exponential w.r.t. n, and on the order hand $\sum_{i=1}^{n} q_i$ must be greater than K and therefore also exponential w.r.t. n, otherwise the constraint would be trivially true and its ROBDD representation would collapse to the $\mathcal{T}$-terminal node.

2.3.3 SAT Encodings of Decision Diagrams

There exist many works in the literature on the encoding of PB constraints into SAT, not only based on BDDs, but also on *adder networks*, *sorting networks* and other approaches [ES06, BBR09, ANO+12, HMS12, TBS13, JMM15]. The most important among these encodings will be revisited and extended in Chapter 7, where we propose generalised versions of them to deal with another constraint that generalises PB constraints.

A recent work comparing different PB SAT encodings is that of [PS15]. That paper introduces the *PBLIB*, a library to translate PB constraints into CNF formulas which includes fifteen different encodings of PB constraints from the literature. In the experiments performed in that paper, the BDD-based approach clearly outperforms the other encodings in terms of solving time.

The use of BDDs to encode PB constraints to SAT was firstly considered

in [ES06]. The approach consists in, first of all, representing the PB constraint as a BDD, then treating the BDD as a circuit of *if-then-else gates*, and finally translating this circuit to clauses by the Tseitin transformation. More precisely, an auxiliary variable has to be added for each node of the BDD, which is constrained to be true with a given assignment of the variables of the PB iff the Boolean function represented for the BDD rooted at this node evaluates to true. Then, for each non-terminal node of the BDD, the following clauses are added:

$$v_f \wedge \overline{x} \rightarrow v \tag{2.9}$$
$$\overline{v_f} \wedge \overline{x} \rightarrow \overline{v} \tag{2.10}$$
$$v_t \wedge x \rightarrow v \tag{2.11}$$
$$\overline{v_t} \wedge x \rightarrow \overline{v} \tag{2.12}$$
$$v_t \wedge v_f \rightarrow v \tag{2.13}$$
$$\overline{v_t} \wedge \overline{v_f} \rightarrow \overline{v} \tag{2.14}$$

where x is the selector of the node, v is the auxiliary variable of the node, and v_t and v_f are the auxiliary variables of the true and false children respectively. Also, the unary clauses

$$r \tag{2.15}$$
$$t \tag{2.16}$$
$$\overline{f} \tag{2.17}$$

need to be added, where r t and f are the auxiliary variables of the root, $\mathcal{T}$-terminal and $\mathcal{F}$-terminal nodes respectively. Therefore this encoding introduces one fresh variable and six ternary clauses per node. Clauses (2.13) and (2.14) are not necessary, but they increase the strength of unit propagation.

At the same time, in [BBR06] an encoding was introduced which, although it is not thought to be derived from an explicit BDD, in essence it is encoding a BDD as proposed in [ES06], with two main differences. The first difference is that the encoding procedure that they proposed cannot guarantee that the implicitly generated BDD is reduced. The second difference is that, by assuming decreasing monotonicity of the PB constraints, it is only required to add clauses (2.15),(2.16),(2.17) as well as the following four clauses for each node,

and there is no loss of consistency nor propagation strength:

$$v_f \wedge \overline{x} \rightarrow v \tag{2.18}$$

$$\overline{v_f} \rightarrow \overline{v} \tag{2.19}$$

$$v_t \rightarrow v \tag{2.20}$$

$$\overline{v_t} \wedge x \rightarrow \overline{v} \tag{2.21}$$

It was however shown later in [ANO+12] that this encoding can be made still much smaller in the case of monotonic PB constraints, since only the following two clauses need to be introduced in addition to clauses (2.15), (2.16), (2.17):

$$\overline{v_f} \rightarrow \overline{v} \tag{2.22}$$

$$\overline{v_t} \wedge x \rightarrow \overline{v} \tag{2.23}$$

This last encoding changes the semantics of the auxiliary variables, since they are only constrained to be false whenever the Boolean function represented by the node evaluates to false, but it is still consistent and maintains GAC by UP.

All the previously presented encodings introduce a number of clauses and fresh variables linear in the size of the ROBDD and therefore can be exponential w.r.t. the number of variables of the PB constraint in the worst case. However it has already been argued in Section 2.3.2 that this is not an issue in most practical applications.

There exist polynomial size encodings of PB constraints [HMS12]. It was proved in [ANO+12] that all PB constraints whose coefficients are powers of two have a polynomial size ROBDD representation. Using the fact that any PB constraint can be reduced to a set of equalities plus a PB constraint whose coefficients are all powers of two, the authors of [ANO+12] provide a GAC polynomial encoding of PB constraints which makes use of BDDs.

Some other related works can been found on SAT encodings of decision diagrams. In [AS14] a generalisation of the two-clause encoding from [ANO+12] is presented. This more generic encoding uses Multi-valued Decision Diagrams (MDD) [SHMB90] to represent Linear Integer Arithmetic (LIA) constraints, and a generalisation of the algorithm in [ANO+12] to construct such MDDs. An MDD is essentially a generalisation of a BDD in which the selectors of the nodes are multi-valued variables, and the nodes have a child for each possible value of the variable. This MDD encoding is revisited in [AGMES16], where also some other encodings of MDDs representing LIA constraints are introduced. These do not try to model the truth value of every node of the diagram, but they introduce auxiliary variables modelling whether an assignment

follows a particular path in the MDD. These path-based encodings introduce fresh variables and clauses both for the nodes and the edges of the MDD, and all of them are of linear size with respect to the size of the MDD.

Chapter 3

LIA and BDD-Based SAT Encodings of Resource Constraints: Application to MRCPSP

The works [ABP+11, Suy13] presented an SMT based system to tackle the RCPSP, concluding that SMT is competitive with state-of-the art solvers for this problem. These results encourage us to tackle other scheduling problems using SMT, and in this chapter we in particular solve the Multi-mode Resource-Constrained Project Scheduling Problem (MRCPSP), which is a generalisation of the RCPSP. This problem is denoted as $MPS|prec|C_{max}$ in [BDM+99] and as $m, 1T|cpm, disc.mu|C_{max}$ in [HDR99].

In the MRCPSP every activity has a number, greater or equal to 1, of execution modes. Each activity mode is described with a pair, formed by the duration of the activity and a vector of resource demands in this mode. Also, in the MRCPSP one distinguishes between renewable resources and non-renewable resources: renewable resources are replenished at each time unit — as in RCPSP—, while for non-renewable ones, resource usage is accumulated across the entire project. For example, a renewable resource could be number of workers, or the capacity of a machine, while a non-renewable resource could be a budget, or some kind of stock. The objective of the MRCPSP is to find a mode and a start time for each activity, such that the makespan is minimised and the schedule is feasible with respect to the precedence and (renewable and non-renewable) resource constraints.

The MRCPSP is NP-hard [BLK83a]. Several exact and heuristic ap-

proaches to solve the MRCPSP have been proposed in recent years. The most common exact approaches for solving this problem are based on branch-and-bound [SD98, ZBY06] and mixed-integer linear programming (MILP) formulations [ZHR08, KKG12, CSE14]. There exists a non-exact hybrid method that uses MILP and SAT [CV11]. In [VLS15] we found an exact hybrid system using large neighbourhood search and failure-directed search which was, as far as we know, the one closing more MRCPSP instances. In [Suy13] it was presented an SMT formulation for the MRCPSP, that we revisit and improve in this chapter.

The purpose of this chapter is twofold. On one hand, we want to study the efficiency of SMT to solve the MRCPSP and provide a comparison with the state of the art of MRCPSP solving. It is worth noting that the MRCPSP contains many disjunctions introduced by the choice of execution modes, and this can be naturally modelled into the SMT language. Moreover, modern SMT solvers are specifically designed to efficiently deal with Boolean combinations of arithmetic predicates. Hence, we can use an off-the-shelf SMT solver without modifying it. On the other hand, we study an encoding approach for resource constraints in SMT alternative to the one presented in [Suy13]. There, the LIA theory was used to handle resource constraints, and now we compare this same approach against another one consisting of using BDD-based SAT encodings of PB constraints. In this second approach the only theory left is IDL, which is easier to decide than LIA. The use of BDD-based encodings of PB constraints has already given good results in solving many Constrained Optimisation Problems [BPSV14]. The two formulations that we present are based on the Time approach described in Section 2.1.3.

A number of preprocessing steps are performed to obtain lower and upper bounds, and to narrow the time windows of the activities. We also use a preprocessing step that allows us to reduce the demand of non-renewable resources for each activity and a simplified method for checking infeasibility. We also define an ad-hoc minimisation procedure, which consists in iteratively running the SMT solver on decisional checks while bounding the cost function, until the optimum is found. This procedure also tries to reformulate the SMT formula by narrowing the time windows of activities after each iterative step as the search advances. The reformulation preserves the internal search state of the SMT solver between successive calls, thus taking advantage of the learning capabilities of the CDCL(T) SMT solvers.

In order to show the efficiency of the presented solution, we report very good results of the experiments conducted on the most challenging MRCPSP instances from the PSPLib [KS97] and on some more challenging instances from MMLIB [VPV14]. Namely, in Section 3.5 we report the results on the j30

set, because that is the only one with open instances in PSPLib and that our system is unable to completely close, and on the MMLIB50 set from MMLIB, which contains harder instances and many of them remain open. We compare our results with [VLS15] which, to the best of our knowledge, was the state-of-the-art of solving MRCPSP when we did this work.

The work presented in this chapter has been published in [BCSV16]. This work is related to the objectives number 1, 2 and 5 of this thesis. The rest of this chapter is organised as follows:

- In Section 3.1 we formally define the MRCPSP.
- In Section 3.2 we describe different preprocessing steps.
- In Section 3.3 we describe our SMT formulations for the MRCPSP.
- In Section 3.4 we describe our optimisation algorithm.
- In Section 3.5 we provide an experimental evaluation of our formulations.
- In Section 3.6 we summarise the contributions of this chapter.

3.1 The Multi-mode Resource-Constrained Project Scheduling Problem

The *Multi-mode Resource-Constrained Project Scheduling Problem* (MRCPSP) is defined by a tuple (V, M, p, E, R, B, b) where :

- $V = \{A_0, A_1, \ldots, A_n, A_{n+1}\}$ is a set of activities. Activities A_0 and A_{n+1} are dummy activities representing, by convention, the start and the end of the schedule, respectively. The set of non-dummy activities is defined by $A = \{A_1, \ldots, A_n\}$.
- $M \in \mathbb{N}^{n+2}$ is a vector of naturals, with M_i being the number of modes that activity A_i can execute in, with $M_0 = M_{n+1} = 1$ and $M_i \geq 1, \forall A_i \in A$.
- p is a vector of vectors of naturals, with $p_{i,o}$ being the duration of activity A_i using mode o, with $1 \leq o \leq M_i$. For the dummy activities, $p_{0,1} = p_{n+1,1} = 0$, and $p_{i,o} > 0, \forall A_i \in A, 1 \leq o \leq M_i$.
- E is a set of pairs of activities representing end-start precedence relations. Concretely, $(A_i, A_j) \in E$ iff the execution of activity A_i must precede

that of activity A_j, i.e., activity A_j must start after activity A_i has finished.

We assume that we are given a precedence activity-on-node graph $G = (V, E)$ that contains no cycles, since otherwise the precedence relation is inconsistent. We assume that E is such that A_0 is a predecessor of all other activities and A_{n+1} is a successor of all other activities.

- $R = \{R_1, \ldots, R_{v-1}, R_v, R_{v+1}, \ldots, R_q\}$ is a set of resources. The first v resources are renewable, and the last $q - v$ resources are non-renewable.
- $B \in \mathbb{N}^q$ is a vector of naturals, with B_k being the available amount of each resource R_k. The first v resource availabilities correspond to the renewable resources, while the last $q - v$ ones correspond to the non-renewable resources.
- b is a three-dimensional matrix of naturals corresponding to the resource demands of activities per mode. Value $b_{i,k,o}$ represents the amount of resource R_k used during the execution of activity A_i in mode o. Note that $b_{0,k,1} = 0$ and $b_{n+1,k,1} = 0, \forall k \in \{1, \ldots, q\}$.

A schedule is a vector of naturals $S = (S_0, S_1, \ldots, S_n, S_{n+1})$ where S_i denotes the start time of activity A_i. We assume that $S_0 = 0$. A schedule of modes is a vector of naturals $SM = (SM_0, SM_1, \ldots, SM_n, SM_{n+1})$ where SM_i, satisfying $1 \leq SM_i \leq M_i$, denotes the mode of each activity A_i. A solution of the MRCPSP problem is a schedule of modes SM and a feasible schedule S of minimal makespan S_{n+1}. The MRCPSP can hence be formally stated as:

Minimise: $$S_{n+1} \tag{3.1}$$

Subject to:

$$(SM_i = o) \rightarrow (S_j - S_i \geq p_{i,o}) \qquad \forall (A_i, A_j) \in E, \forall o \in [1, M_i] \tag{3.2}$$

$$1 \leq SM_i \leq M_i \qquad \forall A_i \in A \tag{3.3}$$

$$\sum_{A_i \in A} \sum_{o \in [1, M_i]} ite\big((SM_i = o) \wedge (S_i \leq t < S_i + p_{i,o}); b_{i,k,o}; 0\big) \leq B_k$$

$$\forall R_k \in \{R_1, \ldots, R_v\}, \forall t \in H \tag{3.4}$$

$$\sum_{A_i \in A} \sum_{o \in [1, M_i]} ite(SM_i = o;\, b_{i,k,o};\, 0) \leq B_k \quad \forall R_k \in \{R_{v+1}, \ldots, R_q\} \quad (3.5)$$

where $ite(c;\, e_1;\, e_2)$ is an *if-then-else* expression denoting e_1 if c is true and e_2 otherwise, $H = \{0, \ldots, UB\}$ is the scheduling horizon, and UB (the length of the scheduling horizon) is an upper bound for the makespan.

A solution is feasible if it satisfies the precedence constraints (3.2), the execution mode correctness constraints (3.3), the renewable resource constraints (3.4) and the non-renewable resource constraints (3.5). An example is shown in Figure 3.1.

A problem instance has no feasible schedules if and only if some of the following conditions holds:

- There exists a cycle in the precedences graph (i.e., an activity is forced to start after it finishes).
- There exists some activity whose demand on a resource in all execution modes is greater than its capacity.
- There is no schedule of modes such that all the non-renewable resource constraints (3.5) are satisfied.

The two first conditions are verifiable in polynomial time. These are typically satisfied in the benchmark instances available in the literature. Hence, having checked them, we propose in Section 3.2.3 to focus on the third condition to detect the infeasibility of an instance.

3.2 Preprocessing

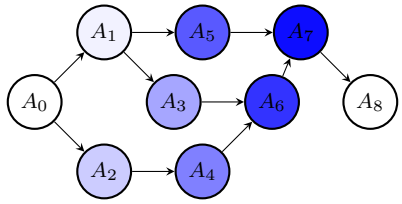
We perform standard preprocessing steps to compute the extended precedence set, a lower and an upper bound for the makespan, and time windows for each activity. We also use a preprocessing technique from [Suy13] called *non-renewable resource demand reduction*, which allows us to reduce the size of the constraints related to non-renewable resources. All these preprocesses are implemented in order to extract some features that will allow us to reduce the size of the SMT formulas and the search space.

3.2.1 Extended Precedence Set

Similarly to the RCPSP, it is possible to compute the transitive closure on the precedence graph to get an extended precedence graph. However, in that

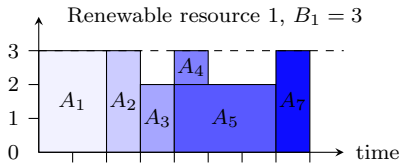
Instance:

	A_0	A_1	A_2	A_3	A_4	A_5	A_6	A_7	A_8
$p_{i,1}$	0	2	4	3	1	3	3	2	0
$b_{i,1,1}$	0	3	1	1	2	2	1	2	0
$b_{i,1,2}$	-	1	1	1	1	1	1	2	-
$p_{i,2}$	-	4	1	1	1	1	2	1	-
$b_{i,2,1}$	0	1	3	2	1	0	0	3	0
$b_{i,2,2}$	-	3	0	1	2	4	2	0	-



Solution:

	A_0	A_1	A_2	A_3	A_4	A_5	A_6	A_7	A_8
S_i	0	0	2	3	4	4	5	7	8
SM_i	1	1	2	2	2	1	2	2	1



Non-renewable resource 2, $B_2 = 8$

$0 + 1 + 0 + 1 + 2 + 1 + 2 + 0 + 0 \leq 8$

Figure 3.1: An MRCPSP instance and its solution. The first table contains the durations and the resource demands (one renewable and one non-renewable) for each activity mode. The graph represents the precedences. The second table contains an optimum schedule *makespan* $= 8$. Finally, the diagram represents the schedule and the validity of the resource usage.

case it has to be taken into account that an activity can have different duration depending on the execution mode. In order to compute the extended precedences, we use the Floyd-Warshall algorithm on the graph defined by the precedence relation E, where each arc (A_i, A_j) is labeled with the duration $\min_{o \in \{1,\ldots,M_i\}}(p_{i,o})$. This extended precedence set is named E^* and contains, for each pair of activities A_i and A_j such that A_i precedes A_j, a tuple of the form $(A_i, A_j, l_{i,j})$ where $l_{i,j}$ is the length of the longest path from A_i to A_j. Note that this longest path length is minimal with respect to the different activity modes. Note also that, if $(A_i, A_i, l_{i,i}) \in E^*$ for some A_i and $l_{i,i} > 0$, then there is a cycle in the precedence relation and therefore the problem instance is inconsistent.

In fact the demands on renewable resources let us go one step further. Note that any activity A_k such that $(A_i, A_k, l_{i,k}) \in E^* \wedge (A_k, A_j, l_{k,j}) \in E^*$ will be completely executed in the time interval $[S_i + \min_{o \in \{1,\ldots,M_i\}}(p_{i,o}), S_j]$. Hence, this interval must be wide enough to run all such A_k without exceeding the availability of any renewable resource. This time interval gives, for every resource $r \in \{1, \ldots, v\}$, a lower bound ($RLB_{i,j,r}$) of the time difference between the end of A_i and the start of A_j:

$$RLB_{i,j,r} = \lceil \frac{1}{B_r} * \sum_{\substack{A_k \in A \\ (A_i,A_k,l_{i,k}) \in E^* \\ (A_k,A_j,l_{k,j}) \in E^*}} \min_{o \in \{1,\ldots,M_k\}} (p_{k,o} * b_{k,r,o}) \rceil$$

So we can update the extended precedence set as:

$$l'_{i,j} = \max(l_{i,j}, \min_{o \in \{1,\ldots,M_i\}}(p_{i,o}) + \max_{r \in \{1,\ldots,v\}}(RLB_{i,j,r})) \quad \forall (A_i, A_j, l_{i,j}) \in E^*$$

where $l_{i,j}$ is the value obtained by transitivity on the precedence set, and $l'_{i,j}$ is the updated value. Note that an increase of a single extended precedence can be propagated to other precedences in E^*. We achieve this propagation with a new execution of the Floyd-Warshall algorithm. This preprocessing resembles the energy based reasoning used by some constraint propagators (see [ADN13]).

3.2.2 Lower Bound

A lower bound LB for the makespan is a lower bound for the start time of activity A_{n+1}. The critical path (i.e. the maximum length path) between the initial activity A_0 and the final activity A_{n+1} in the precedence graph is a lower bound for the makespan. Note that we can easily know the length of this path if we have already computed the extended precedence set, since it corresponds to the value $l_{0,n+1}$ in the tuple $(A_0, A_{n+1}, l_{0,n+1}) \in E^*$.

For instance, in Figure 3.1 the critical path is $[A_0, A_1, A_3, A_6, A_7, A_8]$ with modes $1, 1, 2, 2, 2, 1$, respectively, and its length is 6. Hence, we have $LB = 6$.

Moreover, there exists the possibility that $l'_{0,n+1}$ has been increased due to the renewable resource demands, thus obtaining a better lower bound.

3.2.3 Upper Bound

As stated in Section 2.1.3, it is desired to find a small scheduling horizon $H = \{0, \ldots, UB\}$, since it will have a direct impact on the size of the time windows and therefore the number of variables, the number of constraints and the size of the constraints of our SMT formulas. We compute an upper bound UB for the makespan in two steps.

The first step consists in finding a feasible schedule of modes for the instance, i.e. a schedule of modes that satisfies non-renewable resource constraints (3.5) and the correctness of modes (3.3). This is achieved by a single call to the SMT solver to find a schedule of modes satisfying these constraints. The particular SMT formula that we construct will be indicated in Section 3.3 together with the full problem formulations. As pointed out in Section 3.1, it is a necessary condition that we can find such a feasible schedule of modes, otherwise an instance does not have a feasible solution. If we succeed in this first check, and having tested the other trivial feasibility conditions exposed in Section 3.1, we can conclude that the instance has feasible solutions.

In the second step, given a feasible schedule of modes, we fix the durations and resource demands of the activities according to the schedule of modes, and run the fast greedy PSGS algorithm for the RCPSP explained in Section 2.1.3. It will find a (presumably non-optimal) solution, and we then use its makespan as the upper bound in the scheduling horizon.

3.2.4 Time Windows

We can reduce the domain of each variable S_i (start time of activity A_i), that otherwise would be $\{0\,..\,UB - \min_{o\in\{1,\ldots,M_i\}}(p_{i,o})\}$, by computing its earliest/latest start/close times and time windows similarly to how we explained in Section 2.1.3. It has to be taken into account that the time lags from A_0 and to A_{n+1} are computed as explained in Section 2.1.3:

$$
\begin{aligned}
ES(A_i) &= l_{0,i} \\
EC(A_i) &= ES(A_i) + \min_{o\in\{1,\ldots,M_i\}}(p_{i,o}) \\
LS(A_i) &= UB - l_{i,n+1} \\
LC(A_i) &= LS(A_i) + \min_{o\in\{1,\ldots,M_i\}}(p_{i,o})
\end{aligned}
$$

With these values we can compute the start time window $STW(A_i)$ and run time window $RTW(A_i)$ as explained in Section 2.1 For instance, in the example of Figure 3.1, there is a trivial UB equal to 20, given by the sum of the maximum durations of all the activities. Considering this UB, and having

$l_{4,8} = 4$, the activity A_4 has start time window $[1, 16]$ and run time window $[1, 16]$.

3.2.5 Non-Renewable Resource Demand Reduction

This preprocessing step was presented in [Suy13] and consists in reducing the demand of non-renewable resources in a sound way. As we will see, this will allow us to save SMT literals in the constraints related to those kind of resources.

Let us introduce it through an example. In the example of Figure 3.1, the non-renewable resource R_2 has 8 units available and activity A_6 has two modes: mode 1 requires 1 unit of resource R_2, while mode 2 requires 2 units of the same resource. This problem can be transformed into an equivalent one, where the availability of resource R_2 is 7, and activity A_6 has a demand of 0 units of resource R_2 in mode 1, and of 1 unit in mode 2. Since in mode 1 the demand is of 0 units for this activity, it is not necessary to add any literal considering this mode in the constraints on non-renewable resources. Roughly, following the example, what could be done is to subtract from the availability of resource R_2, and from the different demands of activity A_6 for resource R_2 in each mode, the minimum amount of resource R_2 that activity A_6 needs. However, one could go one step further and, instead of subtracting the minimum demand value, subtract the demand value which most frequently occurs. This of course will lead to negative availabilities and demands. But, interestingly, it allows to reduce the size of the constraints even more (since more demands become zero), while keeping soundness. Details are given below.

For this preprocess, we construct a new vector B' of resource availabilities and a new matrix b' of resource demands. For each non-renewable resource R_k and activity A_i, let $max_{k,i}$ denote the demand value over resource R_k with more occurrences in the different modes of activity A_i (and, in case of a tie, the smallest one). Then we state:

$$b'_{i,k,o} = b_{i,k,o} - max_{k,i} \qquad \forall A_i \in A, \forall o \in \{1, \ldots, M_i\}$$

$$B'_k = B_k - \sum_{A_i \in A} max_{k,i} \qquad \forall R_k \in \{R_{v+1}, \ldots, R_q\}$$

Note that vector B' and matrix b' range now over integers instead of over naturals, i.e., they can contain some negative values. The zero $b'_{i,k,o}$ values —whose number is maximal thanks to the fact that we subtract the demand value with most occurrences— allow us to simplify the constraints on non-renewable constraints, see Equations (3.14) and (3.16) below.

3.3 Formulations

We propose two different SMT formulations for the MRCPSP: *ITE*, which uses the theory of LIA, and *BDD*, which uses the theory of IDL. The difference between the two encodings is in the formulation of the constraints over resources: *ITE* contains summations of *if-then-else* expressions, while *BDD* uses PB constraints which are encoded into CNF using the BDD-based encoding from [ANO+12] described in Chapter 2.

We also introduce some refinements on the encodings considering the preprocessing steps described in Section 3.2 (extended precedences, non-renewable resource demand reduction, etc.)

We introduce the set of integer variables $\{S_0, S_1, \ldots, S_n, S_{n+1}\}$ to denote the start time of each activity. We encode the schedule of modes with the set of Boolean variables $\{sm_{i,o}\,|, 0 \leq i \leq n+1,\ 1 \leq o \leq M_i\}$, with $sm_{i,o}$ being true if and only if activity A_i is executed in mode o.

The objective function (3.1) will be minimised with the iterative process detailed in Section 3.4. We introduce the following constraints in both formulations:

$$S_0 = 0 \tag{3.6}$$

$$S_i \geq ES(A_i) \qquad \forall A_i \in \{A_1, \ldots, A_{n+1}\} \tag{3.7}$$

$$S_i \leq LS(A_i) \qquad \forall A_i \in \{A_1, \ldots, A_{n+1}\} \tag{3.8}$$

$$sm_{i,o} \rightarrow S_j - S_i \geq p_{i,o} \qquad \forall (A_i, A_j) \in E,\ \forall o \in \{1, \ldots, M_i\} \tag{3.9}$$

$$S_j - S_i \geq l_{i,j} \qquad \forall (A_i, A_j, l_{i,j}) \in E^* \tag{3.10}$$

$$\bigvee_{1 \leq o \leq M_i} sm_{i,o} \qquad \forall A_i \in V \tag{3.11}$$

$$\overline{sm_{i,o}} \vee \overline{sm_{i,o'}} \qquad \forall A_i \in V, 1 \leq o < M_i,\ o < o' \leq M_i \tag{3.12}$$

where (3.7) and (3.8) encode the time windows, (3.9) encodes the precedences, (3.10) encodes the extended precedences and (3.11) and (3.12) ensure that each activity runs in exactly one mode.

Constraints (3.4) and (3.5) on resources are differently handled in each of the two formulations, as described below. However, in both cases, for the constraints over renewable resources, we introduce the Boolean variables $x_{i,t}$, which are constrained to be true iff activity A_i is running at time t as follows:

$$\begin{aligned} sm_{i,o} \rightarrow (x_{i,t} \leftrightarrow (S_i \leq t) \wedge (t < S_i + p_{i,o})) \\ \forall A_i \in A, \forall o \in \{1, \ldots, M_i\}, \forall t \in RTW(A_i) \end{aligned} \tag{3.13}$$

3.3.1 *ITE*

This formulation makes use of *if-then-else* (*ite*) expressions in the constraints over resources, which are supported by the theory of LIA. The constraints over the non-renewable resources are the following:

$$\begin{aligned} \sum_{A_i \in A} \sum_{o \in \{1,\ldots,M_i\}} ite\left(sm_{i,o};\, b'_{i,k,o};\, 0\right) \leq B'_k \\ \forall R_k \in \{R_{v+1}, \ldots, R_q\} \end{aligned} \tag{3.14}$$

Notice that the *ite* expression can be removed in the cases where $b'_{i,k,o} = 0$ (recall Section 3.2.5). When using the *ITE* formulation, we will use Constrains (3.14), Constraints (3.11) and Constraints (3.12) to find a schedule of modes —i.e. an assignment on variables $sm_{i,o}$— that let us compute an *UB* as described in Section 3.2.3.

The demands on renewable resources are constrained as follows:

$$\begin{aligned} \sum_{\substack{A_i \in A,\, s.t.: \\ t \in RTW(A_i)}} \sum_{o \in \{1,\ldots,M_i\}} ite\left(sm_{i,o} \wedge x_{i,t};\, b'_{i,k,o};\, 0\right) \leq B'_k \\ \forall R_k \in \{R_1, \ldots, R_v\}, \forall t \in H \end{aligned} \tag{3.15}$$

3.3.2 *BDD*

This formulation expresses the constraints over resources using PB constraints that will be then translated into CNF. We can define the constraints over non-renewable resources as:

$$\begin{aligned} \sum_{A_i \in A} \sum_{o \in \{1,\ldots,M_i\}} b'_{i,k,o} \cdot sm_{i,o} \leq B'_k \\ \forall R_k \in \{R_{v+1}, \ldots, R_q\} \end{aligned} \tag{3.16}$$

Similarly to the *ITE* formulation, note that we can remove from the sum the terms where $b'_{i,k,o} = 0$ to take advantage of the non-renewable resource demand

reduction. When using the *BDD* formulation, we will use Constraints (3.16), Constraints (3.11) and Constraints (3.12) to find an assignment on variables $sm_{i,o}$ that lets us compute a scheduling horizon as described in Section 3.2.3.

To encode the constraints over renewable resources, we are going to define Boolean variables $x_{i,t,o}$ which are true if and only if A_i runs in mode o at time t:

$$x_{i,t,o} \leftrightarrow (sm_{i,o} \wedge x_{i,t}) \qquad \forall A_i \in A,\, \forall t \in RTW(A_i),\; \forall o \in \{1, \ldots, M_i\} \tag{3.17}$$

The constraint over the renewable resources is:

$$\sum_{\substack{A_i \in A,\, s.t.: \\ t \in RTW(A_i)}} \sum_{o \in \{1,\ldots,M_i\}} b'_{i,k,o} \cdot x_{i,t,o} \leq B'_k \qquad \forall R_k \in \{R_1, \ldots, R_v\}, \forall t \in H \tag{3.18}$$

We translate these constraints into CNF using the encoding presented in [ANO+12] and defined in Section 2.3.2 which only introduces two clauses per node of the BDD, but requires the encoded PB to be monotonic decreasing. The monotonicity requirement only lets us use the reformulation suggested in Subsection 3.2.5 if we remove the minimum resource demand over modes of an activity instead of the resource demand occurring more times. If we were removing the resource demand occurring more times, we could get negative resource demands and the PB constraint would contain negative coefficients, resulting in a not monotonic decreasing function.

Notice also that with this encoding of the resources constraints, the only arithmetic predicates remaining in the *BDD* encoding are the ones expressing precedences, namely (3.9) and (3.10), or bounds for integer variables in (3.7), (3.8) and (3.13). Hence, the remaining arithmetic constraints are IDL expressions. In fact, in the *BDD* encoding, all resource constraints are fully controlled by the SAT component of the solver. We argue in the Section 3.5 that this is a key point in the good performance obtained with *BDD*.

3.4 Optimisation

Our system uses the Yices 2.4.2 [Dut14] SMT solver's API to check the satisfiability of the encodings. Yices 2 has shown to be a competitive SMT solver

when considering LIA [DdM06b]. The solving process of our system is presented in Algorithm 4. It basically consists of the following steps:

1. Compute the preprocessed data.
2. Detect infeasibility and end the process, or find a feasible schedule of modes.
3. Use the obtained schedule of modes to compute an UB with PSGS heuristic and narrow the time windows.
4. Find the optimum makespan.

Algorithm 4 solve_MRCPSP

Output: Optimum makespan if feasible. Otherwise return infeasible.

$INS \leftarrow read\_MRCPSP\_instance()$
// INS contains instance data (V, A, M, p, E, R, B, b)
$PREP \leftarrow preprocessing()$
// $PREP$ contains preprocessed data (E^*, ES, LS, B', b')
$ENC \leftarrow encode\_MRCPSP\_SAT(INS, PREP)$
$(SAT, MODEL) \leftarrow smt\_check(ENC)$ // Check feasibility, and give a model if any
if SAT **then**
  $LB \leftarrow l_{0,n+1}$ // Trivial lower bound
  $SM \leftarrow get\_schedule\_of\_modes(MODEL)$
  $UB \leftarrow parallelSGS(INS, SM)$ // Heuristic solution
  $OPT \leftarrow optimise\_feasible(LB, UB, INS, PREP)$
  return OPT
else
  return *INFEASIBLE*
end if

Recall from Section 3.2.3 that we only encode the constraints of correct mode assignment and non-renewable resource constraints for feasibility check (step 2). This is denoted as *encode_MRCPSP_SAT* in Algorithm 4. For the last step of computing the optimum makespan, we have implemented a search procedure, namely *optimise_feasible*, which calls the SMT solver successively constraining the value of the variable S_{n+1} to be smaller or equal to UB. It is described in Algorithm 5. This optimisation algorithm is based on a linear search schema starting from a feasible UB. Every time a feasible schedule is found, UB is updated to take its makespan minus one, and the SMT solver

Algorithm 5 optimise_feasible

Input: LB, UB, feasible instance data (INS), preprocessed data ($PREP$).
Output: Optimum makespan

```
ENC ← encode_MRCPSP(LB, UB, INS, PREP)
smt_assert_encoding(ENC)
SAT ← smt_check()
if SAT then
  MODEL ← smt_get_model()
  MAKESPAN ← smt_get_makespan(MODEL)
  UB ← MAKESPAN − 1
end if
while SAT and UB ≥ LB do
  smt_assert(S_{n+1} ≤ UB) // Bound the makespan
  smt_compress_TW(UB, PREP) // Optional
  SAT ← smt_check()
  if SAT then
    MODEL ← smt_get_model()
    MAKESPAN ← smt_get_makespan(MODEL)
    UB ← MAKESPAN − 1
  end if
end while
if SAT then
  return UB
else
  return UB + 1
end if
```

is called again. This procedure is repeated until we find the biggest infeasible UB. Moreover we compress the time windows at each iterative step of the optimisation process. On the one hand, we assert new single atom clauses of the form $S_i \leq LS(A_i)$ with the updated latest start times for the current UB, therefore bounding the value of S_i. On the other hand, we also assert $(\overline{x_{i,t}})$ for the time instants t that are excluded from the time window of A_i, thus avoiding the solver the work of propagating these values that become trivial. This compression corresponds to the instruction *smt_compress_TW*, and is not mandatory for consistency. However, we explain in Section 3.5 that the compression gives a noticeable speedup in the solving process. Notice that, since we decrease UB linearly, we never find an infeasible UB until the optimum is detected, so that we do not have to retract any constraint and we

can maintain the learning of the SMT solver.

3.5 Results

We have run our experiments on a 8GB Intel® Xeon® E3-1220v2 machine at 3.10 GHz. In all experiments we use Yices 2.4.2 as the core SMT solver, and the timeout is 3600 seconds. Our system is available at the website of the Logic and Programming research group [LAP]. Our executions are made on the j30 [KS97] and MMLIB50 [VPV14] sets of instances. Both sets contain instances with 3 execution modes per activity and 2 renewable and 2 non-renewable resources. There are 552 feasible instances and 88 infeasible instances in j30, each one with 30 activities, and 540 feasible instances of 50 activities in MMLIB50. We provide comparative results with the system presented in [VLS15], which uses a failure directed search for a Constraint Programming optimiser (we refer to this system as *FDS*). To our best knowledge this system had reported the best results of a exact solver for MRCPSP when we ran these experiments.

In both *ITE* and *BDD* settings we will be using the non-renewable resource demand reduction described in Section 3.2.5, since it provides a very noticeable time performance improvement. In Figure 3.2 we illustrate a scenario where this technique has a highly positive impact, that is on the feasible instances of j30 set and for *ITE*. A total of 102 more instances were solved with *ITE* thanks to including this new preprocessing in it. We have also observed an important speedup with *BDD*, although not as high as with *ITE*. Also, looking at Figure 3.2 we can see that there are some instances highly benefited from this preprocessing and some others that are not affected at all. We have observed that the most benefited instances are the ones whose activities have the highest demands on the resources.

In Table 3.1 we analyse the time required to determine the infeasibility of the infeasible instances of j30. We show the performance of simplifying the *ITE* and *BDD* encodings for feasibility checks as explained in Subsection 3.2.3 (i.e. only finding a schedule of modes which satisfies the non-renewable resource constraints). We also include in Table 3.1 the time required if we use the full *ITE* and *BDD* encodings, and finally the time required by *FDS*. It can be seen that our system is better than *FDS* in any case for proving infeasibility, having three orders of magnitude of difference when using the simplified *BDD* encoding. It is specially noticeable the performance improvement achieved by simplifying the encoding, since the computation time is reduced two orders of magnitude both for *ITE* and *BDD*.

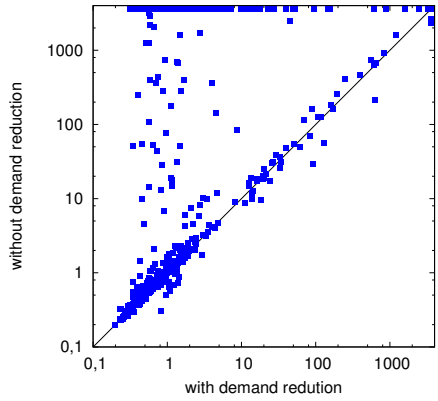


Figure 3.2: Comparison of solving time (in seconds) with the *ITE* encoding using the non-renewable resource demand reduction (axis x) and without using it (axis y) on the j30 set.

solver	**25%**	**median**	**75%**	**max**	**mean**	**solved**
ITE simp.	0.11	0.2	0.52	8.83	0.57	88
ITE full	7.55	14.6	28.44	416.68	27.49	88
BDD simp.	0.03	0.05	0.08	0.58	0.09	88
BDD full	1.39	1.97	2.75	5.63	2.23	88
FDS	27.15	53.07	107.12	462.7	91.48	88

Table 3.1: Solving times in seconds and number of instances solved of the infeasible instances of j30 set.

We have evaluated on the feasible instances of j30 how invoking the procedure *smt_compress_TW* of Algorithm 5 helps to boost the solving process. Using the compression in *ITE* lets us solve 4 more instances than not using it, and the solving time is in average reduced a 55.22% (considering only the instances solved in both cases). Regarding the *BDD* encoding, we have observed that this compression does not suppose a clear improvement of the solving time as happens with *ITE* (neither a worsening), what suggests that *BDD* propagates better the implications of reducing the upper bound.

Figure 3.3 reflects the performance differences between *ITE* and *BDD*, based on the satisfiable instances of j30 and the instances of MMLIB50. The computation of the Binary Decision Diagrams requires some time, and it is penalising the overall performance of the easiest to solve instances, which makes *ITE* clearly the best option for the easiest instances. Regarding the hardest instances, *BDD* supposes an important speedup, and it is able to solve

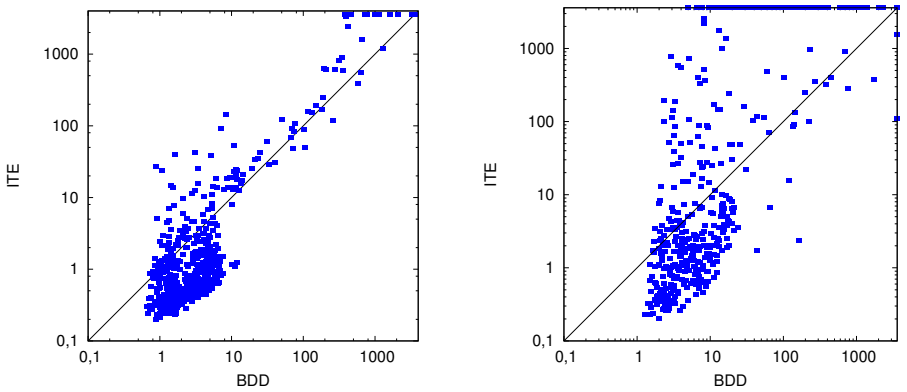


Figure 3.3: Comparison of solving time (in seconds) with the *ITE* encoding and the *BDD* encoding for the feasible instances of j30 set (left) and MMLIB50 set (right).

set	solver	Q1	median	Q3	mean	solved
j30 (feasible)	**ITE**	0.50	0.91	2.25	141.92	535
	BDD	1.42	2.80	5.08	89.48	544
	FDS	0.02	0.04	0.86	98.17	543
MMLIB50	**ITE**	1.37	7.56	timeout	1251.03	359
	BDD	3.93	9.87	155.79	692.17	445
	FDS	0.05	1.34	1028.94	894.20	415

Table 3.2: Solving times in seconds and number of instances solved of the feasible instances of j30 set and of MMLIB50 set. The unsolved instances have been counted as 3600 seconds.

9 more instances than *ITE*. An indicator of the performance of the encodings is the number of conflicts encountered during the solving process. With *ITE*, the average number of conflicts encountered by the theory solver during the last optimisation iteration was 31964, while with *BDD* was 216 (only considering the instances solved in both cases). It is not surprising that *BDD* has few theory conflicts, since the constraints over resources are fully encoded with Boolean formulas. But despite this fact, the number of Boolean conflicts is also significantly smaller with *BDD* (24813) compared to *ITE* (87227), which may suggest that with *BDD* the lemmas learned from the conflicts achieve a better prune of the search space. It can be seen in Table 3.2 how the advantage of *BDD* in front of *ITE* is still greater in MMLIB50 set.

Table 3.2 also contains results comparing our system and *FDS* in j30 and MMLIB50 sets. It can be seen that, although *FDS* goes faster in the easiest

instances, we scale better using *BDD* and are able to solve more instances than them in both sets. In j30, we solve one instance more than *FDS* (only 8 remain unsolved), and the mean solving time is slightly smaller. But the advantage of *BDD* is specially noticeable in MMLIB50, which is the hardest set, having a third quartile around one order of magnitude smaller, a mean solving time a 22,6% smaller, and solving 30 more instances.

Looking at each instance in particular in j30, *BDD* closes 2 instances that *FDS* is unable to solve within the given timeout, and *FDS* solves one that *BDD* does not. Regarding MMLIB50, the best results reported until now were achieved by [Gei13] using metaheuristics. With *BDD*, which is an exact method, we have been able to certify the optimality of 445 instances, and we have improved the upper bound of a total of 51 instances with respect to [Gei13]. On the other hand, the number of instances solved by *BDD* and not solved by *FDS* is 33, and 3 instances are solved by *FDS* and not by *BDD*.

3.6 Chapter Summary

We have shown in this chapter that SMT is a competitive approach for the MRCPSP. With some classical and a new preprocessing methods, using two SMT encodings, and using an off-the-shelf SMT solver as a decision procedure, we have developed a robust and exact MRCPSP solver. The principal preprocessing methods that we use are a thorough computation of time windows, a reduction of the size of non-renewable resource constraints, and a pre-solving check of feasibility that also let us rapidly compute an upper bound of the makespan. Our results show that a combination of Boolean encoding for constraints over resources and IDL encoding for constraints over precedences is specially competitive in solving hard instances. It is worth noting that SMT provides not only efficiency, but also an expressive language for this kind of problems.

Chapter 4

Using Collateral Constraints to Compactly Encode PB Constraints to SAT

PB constraints appear frequently in formulations of CSP. Sometimes, there are also other constraints imposed on the Boolean variables of a PB constraint. A frequent case is that of the AMO constraint, which states that at most one of the Boolean variables in a set can be assigned true. For example, in routing problems [MTZ60, DR59, Lap92], the length of paths can be represented using PB constraints, where each variable encodes if the path is joining two particular points. Since it is usually required that Hamiltonian paths are followed, only one variable among the ones which represent going from a particular point to any other can be set to true. Also, in combinatorial auctions the objective function is usually a PB constraint [DVV03, BBMV13], where each Boolean variable represents whether a certain bid has been selected. The PB constraint contains all the possible bids, but many bids can contain a same product, and therefore at most one among them can be selected. In the general context of scheduling, PB constraints are a natural way to express constraints over the use of shared resources. An example is the MRCPSP formulation provided in Chapter 3, where many notions of incompatibility between activities arise: there can be precedences between activities, and must be chosen a single running mode for each activity. In all the example problems mentioned it is also usual to find EO constraints, in which case assigning all variables to false is also disallowed.

It is well known that the search for small SAT encodings is a useful technique to achieve better solving times [BW03, EB05]. As we explained in Sec-

tion 2.3.3, a commonly used approach is to represent a PB constraint as a BDD, and then encode the BDD to SAT. Finding small BDD representations is crucial, since the size of the generated SAT formula is proportional to the size of the BDD. In this chapter we show how to reduce the size of the decision diagram representations of PB constraints, taking into account the AMO constraints that are also present (either explicitly or implicitly) in the problem. The overall idea is to remove from the decision diagrams the paths whose corresponding variable assignments are already forbidden by the AMO constraints. This way, the decision diagrams do not represent such inconsistent assignments, i.e., they only cover the subset of the assignments that are consistent with the AMO constraints, and hence the decision diagrams can be much smaller.

In summary, we propose to obtain small SAT encodings of PB constraints by taking into account that some assignments are forbidden due to other constraints. We introduce the general notion of PB($\mathcal{C}$) constraint, which is defined as the conjunction of a PB constraint with a set of constraints $\mathcal{C}$. In particular, we focus on PB(AMO) constraints, defined as the conjunction of a PB constraint and a set of AMO constraints. We use a compact decision diagram representation for the PB constraints, taking into account the AMO constraints imposed. Finally, by encoding such compact diagrams to SAT, together with the rest of constraints, we are able to obtain very small SAT encodings of PB(AMO) constraints. As we show in the experimental section, the small size of the encodings obtained with our treatment has a dramatic impact on the solving time of some well-known problems.

A preliminary version of the work presented in this chapter was published in [BCSV17b]. There, we introduced the PB(AMO) technique slightly differently than how is presented here, and also explained how to use this approach to encode scheduling problems. In this thesis we have decoupled the technique of PB(AMO) from its particular application to scheduling. First, in this chapter we introduce all the concepts related to PB(AMO) and some theoretical properties, and evaluate the performance of this technique on the decisional version of scheduling problems without entering into formulation details. Then, how to apply this technique in scheduling problems will be explained in more detail in Chapters 5 and 6.

This chapter is related to the objectives 3 and 4 of this thesis. The rest of the chapter is organised as follows:

- In Section 4.1 we introduce PB($\mathcal{C}$) constraints and the particular case of PB(AMO) constraints.
- In Section 4.2 we present AMO-MDDs, a Multi-valued Decision Diagram

representation for PB constraints under the assumption of AMO constraints, and provide an algorithm to construct reduced ordered AMO-MDDs.

- In Section 4.3 we provide an AMO-MDD based SAT encoding of monotonic decreasing PB(AMO) constraints. We prove its correctness and that it UP-maintains GAC.

- In Section 4.4 we present some reformulation techniques that allow us to reduce other PB($\mathcal{C}$) constraints to monotonic decreasing PB(AMO) constraints, namely (i) PB(EO) constraints; (ii) not monotonic decreasing PB(AMO) constraints; (iii) PB constraints with implication chains.

- In Section 4.5 we provide an experimental evaluation of the impact of the presented technique on the size of the encodings and on solving time. contributions.

- In Section 4.6 we summarise the contributions of this chapter.

4.1 Conjunctions of Pseudo-Boolean Constraints with Other Constraints

Given a constraint $\mathcal{P}$ of the form $P \wedge C_1 \wedge \cdots \wedge C_m$, where P is a PB constraint and $C_1, \ldots, C_m$ are any other constraints, a straightforward approach to encode it is to generate a formula $E(P) \wedge E(C_1 \wedge \cdots \wedge C_m)$, where $E(P)$ is an encoding of P, and $E(C_1 \wedge \cdots \wedge C_m)$ is an encoding of $C_1 \wedge \cdots \wedge C_m$. We propose to relax the encoding of P by only considering assignments that satisfy $C_1 \wedge \cdots \wedge C_m$, since the remaining assignments falsify $\mathcal{P}$.

Let us develop this idea with a motivating example.

Example 3. *Consider a constraint $\mathcal{P} : P \wedge C_1 \wedge C_2$ over Boolean variables x_1, x_2, x_3. Let us suppose that $\mathcal{P}$, P, C_1 and C_2 have the following truth table, with all the possible assignments labelled $A_0, \ldots, A_7$:*

	$\mathbf{x_1}$	$\mathbf{x_2}$	$\mathbf{x_3}$	$\boldsymbol{P}$	$\boldsymbol{C_1}$	$\boldsymbol{C_2}$	$\boldsymbol{P \wedge C_1 \wedge C2}$
A_0	*0*	*0*	*0*	*0*	*1*	*0*	*0*
A_1	*0*	*0*	*1*	*1*	*1*	*1*	*1*
A_2	*0*	*1*	*0*	*1*	*1*	*0*	*0*
A_3	*0*	*1*	*1*	*0*	*0*	*1*	*0*
A_4	*1*	*0*	*0*	*1*	*1*	*1*	*1*
A_5	*1*	*0*	*1*	*0*	*0*	*0*	*0*
A_6	*1*	*1*	*0*	*0*	*1*	*1*	*0*
A_7	*1*	*1*	*1*	*1*	*0*	*1*	*0*

The straightforward approach to encode the constraint is to generate two formulas $E(P)$, $E(C_1 \wedge C_2)$ which are encodings of P and $C_1 \wedge C_2$ respectively, so that $E(P) \wedge E(C_1 \wedge C_2)$ is an encoding of $\mathcal{P}$. Note that the assignments A_0, A_2, A_3, A_5 and A_7 are not extendable to a model of $E(C_1 \wedge C_2)$, regardless of the satisfiability of $E(P)$ under these assignments. Therefore, $E(P)$ could be replaced by a formula F such that A_1 and A_4 are extendable to a model of F, and A_6 is not. For such formula F, independently of its evaluation on assignments A_0, A_2, A_3, A_5 and A_7, we have that $F \wedge E(C_1 \wedge C_2)$ is an encoding of $\mathcal{P}$. The potential of this idea is that F can be substantially smaller than $E(P)$, and this can have a positive impact on the solving time.

Definition 4.1.1. *Let P be a PB constraint and $\mathcal{C} = \{C_1, \ldots, C_m\}$ be a set of constraints over the variables of P. We will refer to the formula $P \wedge C_1 \wedge \cdots \wedge C_m$ as a $PB(\mathcal{C})$ constraint, and call it a* PB modulo $\mathcal{C}$ *constraint.*

Following the idea of Example 3, we propose to encode PB($\mathcal{C}$) constraints in a combined way. On the one hand we will encode the conjunction of constraints in $\mathcal{C}$ in the usual way, i.e., by encoding each of them separately and using the conjunction of all the resulting clauses. On the other hand, we will translate the PB constraint P into set of clauses that is equisatisfiable assuming that the accompanying constraints $\mathcal{C}$ are already enforced. This way, the set of clauses for P can be significantly smaller than an encoding of the PB constraint alone.

The following lemma trivially follows from the definition of encoding (see Chapter 2):

Lemma 4.1.2. *Let $P \wedge C_1 \wedge \cdots \wedge C_m$ be a PB($\mathcal{C}$) constraint, and $E(C_1 \wedge \cdots \wedge C_m)$ an encoding of $C_1 \wedge \cdots \wedge C_m$. Let F be a formula such that any assignment A satisfying $C_1 \wedge \cdots \wedge C_m$ can be extended to a model of F iff $A \models P$. Then, $F \wedge E(C_1 \wedge \cdots \wedge C_m)$ is an encoding of $P \wedge C_1 \wedge \cdots \wedge C_m$.*

We can go a bit further in the idea of encoding a relaxation of a pseudo-Boolean constraint. In the context of a bigger formula, some constraints $\mathcal{C}$ can

be logically implied. We can take into account those implied constraints $\mathcal{C}$ to relax the encoding of a PB constraint. Moreover, there is no need to encode the implied constraints.

Lemma 4.1.3. *Let P be a PB constraint, B any Boolean function and $E(B)$ an encoding of B. Let $\mathcal{C} = \{C_1, \ldots, C_m\}$ be a set of constraints such that $B \models C_1 \wedge \cdots \wedge C_m$. Let F be a formula such that any assignment A satisfying $C_1 \wedge \cdots \wedge C_m$ can be extended to a model of F iff $A \models P$. Then, $F \wedge E(B)$ is an encoding of $P \wedge B$.*

In this work we focus in the encoding PB(AMO) constraints, a particular case of PB($\mathcal{C}$) constraints defined as the conjunction of a PB constraint and a set of AMO constraints.

Definition 4.1.4. *By* PB(AMO) constraint *we refer to a constraint of the form $P \wedge M_1 \wedge \cdots \wedge M_m$, where P is a PB constraint, and $M_1, \ldots, M_m$ are AMO constraints.*

We will assume that the AMO constraints in a PB(AMO) constraint have disjoint scopes, and that the scope of $M_1 \wedge \cdots \wedge M_m$ is the same as the scope of P, in other words, that $\{scope(M_1), \ldots, scope(M_m)\}$ is a partition of $scope(P)$.

Note that a variable x can always be included in a single-variable AMO constraint of the form $x \leq 1$. Therefore, PB constraints are a particular case of PB(AMO) constraints.

4.2 MDD-Based Representation of PB Constraints with AMO Relations

In this section we show how to represent PB constraints under the assumption of AMO constraints using Multi-valued Decision Diagrams (MDD). In their classical definition, MDDs can be seen as a generalization of BDDs which have a multi-valued selector variable in each node instead of just a Boolean variable [SHMB90], and each possible value corresponds to a different decision. However, and especially in the context of SAT encodings of MDDs, a set of Boolean variables can be used as selectors, each variable representing a different decision. We introduce the following variant of MDD.

Definition 4.2.1. *An* At-Most-One Multi-Decision Diagram *(AMO-MDD) is a generalisation of a BDD. It is a rooted, directed, acyclic multigraph which has two terminal nodes, namely $\mathcal{F}$-*terminal *and $\mathcal{T}$-*terminal*. Each non-terminal node has associated a set of Boolean selector variables $x_1, \ldots, x_l$ and has an*

outgoing edge for each variable. Moreover, there is an additional outgoing edge, which we denote as the else *edge. Each one of the* $l+1$ *outgoing edges corresponds to a different decision, namely assigning exactly one of the* x_i *to true, for* $i \in 1..l$*, or assigning all of them to false, hence choosing the* else *edge. The definitions of ordered and reduced can be also applied to AMO-MDDs, producing AMO-ROMDDs.*

Given a PB(AMO) constraint of the form $P \wedge M_1 \wedge \cdots \wedge M_m$, such that $X_i = \mathit{scope}(M_i)$ for $i \in 1..m$ and $\{X_1, \ldots, X_m\}$ is a partition of $\mathit{scope}(P)$, we will represent its PB constraint P by an AMO-MDD where X_i will be the set of selectors of the nodes in layer i. This way, all paths from the root to a terminal node will choose (assign to 1) at most one of the variables in the scope of each M_i and, therefore, the AMO-MDD will cover all the assignments that satisfy $M_1 \wedge \cdots \wedge M_m$. In this representation, every one of those truth assignments will follow a path from the root to the $\mathcal{T}$-terminal if it satisfies P, or to the $\mathcal{F}$-terminal otherwise. Note that assignments which do not satisfy $M_1 \wedge \cdots \wedge M_m$ (i.e., assigning more than one variable to 1 in the scope of some M_i) are not represented in this AMO-MDD.

As said, unlike ROBDDs representing PB constraints where the i-th layer deals with a single variable x_i, the i-th layer of an AMO-ROMDD deals with a set of variables X_i. Hence, the order of the AMO-ROMDD is defined on sets of selector, i.e., the AMO-ROMDD representation is subject to an ordered partition of the variables of the PB constraint. Figure 4.1 shows the AMO-ROMDD representation of $P : 2x_1 + 3x_2 + 4x_3 + 6x_4 \leq 7$ with the ordered partition $\{x_1, x_2\} \prec \{x_3, x_4\}$. Notice the significant reduction in size with respect to the respective ROBDD representation of P. In particular, the AMO-ROMDD has only 2 non-terminal nodes and 6 edges instead of the 6 non-terminal nodes and 12 edges of the ROBDD, and the number of layers is reduced from 5 to 3.

4.2.1 AMO-ROMDD Construction

[ANO+12] introduced an algorithm to construct a ROBDD representing a PB constraint with a given order, whose running time is polynomial w.r.t. the size of the resulting ROBDD. Here we present a generalisation of that algorithm to construct an AMO-ROMDD for a PB constraint with a given ordered partition $\mathcal{X}$ of its variables. Before describing the algorithm, we adapt the idea of interval for a PB constraint with an associated partition $\mathcal{X}$.

Definition 4.2.2. *Let* $P : \sum_{i=1}^{n} q_i x_i \leq K$ *be a PB constraint and let* $\mathcal{X}$ *be a partition of its variables. The* interval *of* P *with the partition* $\mathcal{X}$ *includes all the*

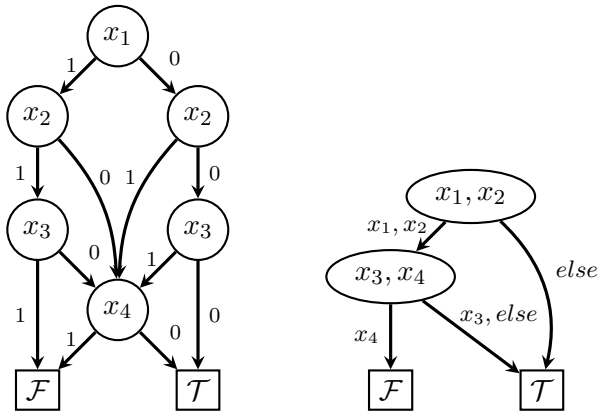


Figure 4.1: Left: ROBDD for the PB constraint $2x_1+3x_2+4x_3+6x_4 \leq 7$ with order $x_1 \prec x_2 \prec x_3 \prec x_4$. Right: AMO-ROMDD for $2x_1+3x_2+4x_3+6x_4 \leq 7$, with ordered partition $\{x_1, x_2\} \prec \{x_3, x_4\}$.

integers K' such that $\sum_{i=1}^{n} q_i x_i \leq K'$, interpreted as a Boolean function, has the same evaluation as P for any assignment A such that $A \models \sum_{x_j \in X_i} x_j \leq 1$, for all $X_i \in \mathcal{X}$. The set of such K' is always an interval that we denote by $[\beta, \gamma]$, as also happens with intervals of PB constraints [ANO+12]. The difference is that in our case the equivalence among the values of the interval only holds on the set of assignments satisfying the AMOs.

Example 4. *The interval of the PB constraint $x_1 + 5x_2 + 4x_3 + 4x_4 \leq 6$ with the partition $\{\{x_1, x_2, x_3\}, \{x_4\}\}$ is $[5, 7]$, because its truth table, limited to the assignments satisfying $x_1 + x_2 + x_3 \leq 1$ and $x_4 \leq 1$, is the same for any $K' \in [5, 7]$, and different for $K' = 4$ and $K' = 8$.*

The AMO-ROMDD representation for a given PB constraint and ordered partition of its variables is the same for any K' in its interval. Note also that every node of an AMO-ROMDD is the root of an AMO-ROMDD, and hence it also has its corresponding interval. In particular, every node at layer i is the root of an AMO-ROMDD representing the PB constraint $\sum_{j=i}^{m} \sum_{x_k \in X_j} q_k x_k \leq K'$ with ordered partition $X_i \prec \cdots \prec X_m$, with a different K' at each node. The algorithm presented below maintains a set L_i of tuples of the form $([\beta, \gamma], \mathcal{M})$ for each layer i of the AMO-ROMDD, where $\mathcal{M}$ is an AMO-ROMDD and $[\beta, \gamma]$ is its corresponding interval. By means of dynamic programming, the AMO-ROMDD representation of a particular PB is constructed only once, it is inserted in L_i together with its interval, and it is reused as the representation of any other PB of the same layer with the same interval.

Algorithm 6 Construction of AMO-ROMDD algorithm

Input: list $\Pi : p_1, \ldots, p_m$, integer K
Output: returns $\mathcal{M}$ the AMO-ROMDD of (Π, K)

1: **for all** $i \in 1..m+1$ **do**
2: $\quad L_i \leftarrow \left\{ ((-\infty, -1], \mathcal{F}), ([\sum_{j \in i..m} \max_{q_k x_k \in p_j} (q_k), \infty), \mathcal{T}) \right\}$
3: **end for**
4: $\mathcal{L} \leftarrow \langle L_1, \ldots, L_{m+1} \rangle$
5: $([\beta, \gamma], \mathcal{M}) \leftarrow \textbf{MDDBuild}(1, \Pi, K, \mathcal{L})$
6: **return** $\mathcal{M}$

Algorithm 7 Procedure MDDBuild

Input: integer $i \in [1, m+1]$, list $\Pi : p_i, \ldots, p_m$, integer K,
input/output list $\mathcal{L}$
Output: returns $[\beta, \gamma]$ interval of (Π, K), and $\mathcal{M}$ its AMO-ROMDD

1: $([\beta, \gamma], \mathcal{M}) \leftarrow \textbf{search}(K, L_i)$
2: **if** $[\beta, \gamma] \neq \emptyset$ **then**
3: $\quad$ **return** $([\beta, \gamma], \mathcal{M})$
4: **else**
5: $\quad$ **let** $p_i = q_1 x_1, \ldots, q_l x_l$
6: $\quad$ **for all** $j \in 1..l$ **do**
7: $\quad\quad ([\beta_j, \gamma_j], \mathcal{M}_j) \leftarrow \textbf{MDDBuild}(i+1, \langle p_{i+1}, \ldots, p_m \rangle, K - q_j, \mathcal{L})$
8: $\quad$ **end for**
9: $\quad ([\beta_{else}, \gamma_{else}], \mathcal{M}_{else}) \leftarrow \textbf{MDDBuild}(i+1, \langle p_{i+1}, \ldots, p_m \rangle, K, \mathcal{L})$
10: $\quad \alpha \leftarrow \textbf{argmax}(q_1, \ldots, q_l)$
11: $\quad$ **if** $[\beta_\alpha, \gamma_\alpha] = [\beta_{else}, \gamma_{else}]$ **then**
12: $\quad\quad$ // This is a long edge
13: $\quad\quad \mathcal{M} \leftarrow \mathcal{M}_\alpha$
14: $\quad\quad [\beta, \gamma] \leftarrow [\beta_\alpha + q_\alpha, \gamma_\alpha]$
15: $\quad$ **else**
16: $\quad\quad$ // This is a new node
17: $\quad\quad \mathcal{M} \leftarrow \textbf{mdd}(\langle x_1, \ldots, x_l \rangle, \langle \mathcal{M}_1, \ldots, \mathcal{M}_l \rangle, \mathcal{M}_{else})$
18: $\quad\quad [\beta, \gamma] \leftarrow [\beta_{else}, \gamma_{else}] \cap \bigcap_{j \in 1..l} [\beta_j + q_j, \gamma_j + q_j]$
19: $\quad$ **end if**
20: $\quad$ **insert**$(([\beta, \gamma], \mathcal{M}), L_i)$
21: $\quad$ **return** $([\beta, \gamma], \mathcal{M})$
22: **end if**

The main procedure is described in Algorithm 6. In order to simplify the notation of recursive calls, the algorithm receives as input a PB constraint $P : \sum_{i=1}^{n} q_i x_i \leq K$ and an ordered partition $X_1 \prec \cdots \prec X_m$ of its variables, represented by a pair (Π, K). Here, $\Pi = p_1, \ldots, p_m$ is a list of sets of monomials, such that every set p_i contains the monomials $q_j x_j$ for each variable $x_j \in X_i$, $i \in 1..m$. The algorithm starts by inserting the $\mathcal{T}$-terminal and the $\mathcal{F}$-terminal AMO-ROMDDs to every layer, with the corresponding interval in that layer. Then, it calls the recursive procedure *MDDBuild* (Algorithm 7). Figure 4.2 shows an example of the contents of each L_i after the construction of an AMO-ROMDD.

Algorithm 7 uses the following functions:

`search`(K, L_i)**:** If there is a tuple $(I, \mathcal{M})$ in L_i, such that $K \in I$, it is returned. Otherwise, an empty interval is returned in the first component of the tuple.

`insert`$((I, \mathcal{M}), L_i)$**:** Inserts $(I, \mathcal{M})$ into the set L_i.

`argmax`$(q_1, \ldots, q_l)$**:** Returns the index of the maximum coefficient in the list $q_1, \ldots, q_l$.

`mdd`$(\langle x_1, \ldots, x_l \rangle, \langle \mathcal{M}_1, \ldots, \mathcal{M}_l \rangle, \mathcal{M}_{else})$**:** Constructs an AMO-ROMDD with a new node as root, $\mathcal{M}_j$ as child for each selector variable x_j, and $\mathcal{M}_{else}$ as the *else* child.

The algorithm in [ANO+12] runs in polynomial time with respect to the size of the generated ROBDD. We argue that Algorithm 6, which is a generalisation for the case of AMO-ROMDDs, preserves the polynomial running time. All the searches and insertions in $\mathcal{L}$ in Algorithm 7 can be done in logarithmic time. Algorithm 7 is called once for the root node of the AMO-ROMDD, and $\mathcal{O}(h \cdot l)$ times for each edge of the AMO-ROMDD, being h the length of the edge, and $l = \max_{p_i \in \Pi} |p_i|$.

4.3 An AMO-MDD based SAT Encoding of Monotonic Decreasing PB(AMO) Constraints

In this section we present an adaptation of the encoding for monotonic decreasing functions from [ANO+12] presented in Section 2.3.3 which is the smallest known BDD-based encoding and UP-maintains GAC on monotonic decreasing PB constraints. Our adaptation deals with PB(AMO) constraints $\mathcal{P} : P \wedge M_1 \wedge \cdots \wedge M_m$, where P is a monotonic decreasing PB constraint,

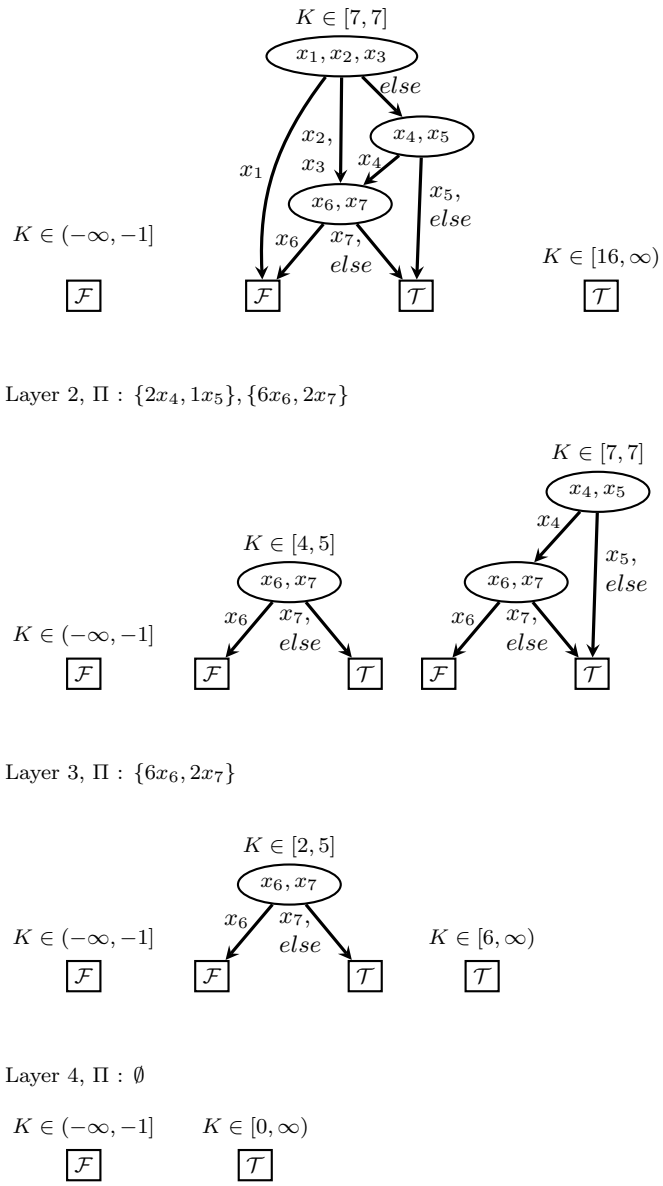


Figure 4.2: Content of $\mathcal{L}$ at the end of the construction of the AMO-ROMDD for $8x_1 + 2x_2 + 3x_3 + 2x_4 + x_5 + 6x_6 + 2x_7 \leq 7$ with the ordered partition $\{x_1, x_2, x_3\} \prec \{x_4, x_5\} \prec \{x_6, x_7\}$.

i.e. P is of the form $\sum_{i=1}^{n} q_i x_i \leq K$, with positive coefficients. We propose a method for obtaining a set of clauses F for P such that in conjunction with an encoding of $M_1 \wedge \cdots \wedge M_m$ gives us an encoding of $\mathcal{P}$.

Though the generated set of clauses F is not exactly an encoding of P, we refer to it as the *Minimal Encoding*, due to its resemblance to the MDD encoding for Linear Integer expressions presented in [AGMES16], also named Minimal. Corollary 4.3.2 below gives the exact Boolean function encoded by the Minimal Encoding.

Minimal Encoding. The encoding adds a fresh auxiliary Boolean variable v for each node of the AMO-MDD $\mathcal{M}$ at hand. If the root of $\mathcal{M}$ is the $\mathcal{T}$-terminal node (i.e., it represents a tautology) the encoding only adds the clause v_r, with v_r being the auxiliary variable of the root node. On the other hand, if the root of $\mathcal{M}$ is the $\mathcal{F}$-terminal node (i.e., it represents a contradiction) the encoding adds the clauses v_r and $\overline{v_r}$. Finally, if the root of $\mathcal{M}$ is not a terminal node, the clauses v_r, $v_\mathcal{T}$ and $\overline{v_\mathcal{F}}$ are included in the formula, where $v_r, v_\mathcal{T}, v_\mathcal{F}$, are the auxiliary variables of the root, the $\mathcal{T}$-terminal and the $\mathcal{F}$-terminal nodes respectively, and the following clauses are added for each non-terminal node with set of selector variables X_i:

$$v_j \vee \overline{x_j} \vee \overline{v} \qquad \forall x_j \in X_i \text{ s.t. } v_j \neq v_0 \tag{4.1}$$

$$v_0 \vee \overline{v} \tag{4.2}$$

Here v is the auxiliary variable of the node, v_j is the auxiliary variable of the child node corresponding to selector x_j, and v_0 is the auxiliary variable of the *else* child. Notice that a node can have more than one selector variable pointing to the same child. In particular, if the edge of a selector x_j points to the *else* child, there is no need to add the clause for the selector variable, because $v_0 \vee \overline{v}$ implies $v_0 \vee \overline{x_j} \vee \overline{v}$.

Theorem 4.3.1. *Let $P : q_1 x_1 + \cdots + q_n x_n \leq K$ be a monotonic decreasing PB constraint and $X = \{X_1, \ldots, X_m\}$ be a partition of its variables. Let F be the Minimal Encoding of an AMO-MDD representation for P and X, and v_r the auxiliary variable of its root node. Let A be any total assignment over $\{x_1, \ldots, x_n\}$, and let G denote the constraint $\sum_{i=1}^{m} \max_{x_j \in X_i} (q_j \cdot x_j) \leq K$. Then the following holds:*

- *If A is a model of G, then A can be extended to a model of F.*
- *If A is not a model of G, then there exists an extension of A that satisfies $F \setminus \{v_r\}$, and any such extension sets v_r to* false.

Proof. We proceed by induction on the number of summands of G. Note that G has one summand for each set X_i.

Base case: In the base case, G is of the form $0 \leq K$, and so is P.

- Assume that we have an assignment A over the variables of P (in fact, an empty assignment) satisfying G, i.e., $K \geq 0$. In this case, the AMO-MDD representation for P and X is simply the $\mathcal{T}$-terminal node. Hence, its Minimal Encoding F contains only the unit clause v_r, and $A \cup \{v_r\}$ is trivially a model of F.
- Assume the contrary, i.e., that we have an assignment A over the variables of P not satisfying G, i.e., A is empty and $K < 0$. In this case, the AMO-MDD representation of P and X is simply the $\mathcal{F}$-terminal node. Hence, its Minimal Encoding F consists of the two unit clauses $v_r, \overline{v_r}$. In this case, $A \cup \{\overline{v_r}\}$ is the only extension of A satisfying $F \setminus \{v_r\}$.

Inductive step: In this case G is of the form $\sum_{i=1}^{m} \max_{x_j \in X_i} (q_j \cdot x_j) \leq K$, with $m \geq 1$, and P can be written as $\sum_{i=1}^{m} \sum_{x_j \in X_i} q_j x_j \leq K$.

Let $\mathcal{M}$ be an AMO-MDD representation for P and X, and F its Minimal Encoding. Let X_1 be of the form $\{x_1, \ldots, x_s\}$ and assume, w.l.o.g., that $q_i \leq q_{i+1}$ for all $i \in \{1, \ldots, s-1\}$.

In order to ease the proof, we define $q_0 = 0$, a neutral coefficient for the *else* case, and introduce the Boolean constant $x_0 = true$, which we assume belongs to any assignment, and allows us to replace the clause $v_0 \vee \overline{v_r}$ in F by $v_0 \vee \overline{x_0} \vee \overline{v_r}$.

Since all coefficients in P are positive, we know that $q_0 \leq q_1$. For all $k \in \{0, \ldots, s\}$, we define the following terms:

- P_k is $\sum_{i=2}^{m} \sum_{x_j \in X_i} q_j x_j \leq K - q_k$, i.e., the constraint resulting of assigning $x_k = true$ in P.
- G_k is $\sum_{i=2}^{m} \max_{x_j \in X_i} (q_j \cdot x_j) \leq K - q_k$, i.e., the constraint resulting of assigning $x_k = true$ in G.
- F_k is the Minimal Encoding for the subgraph of $\mathcal{M}$ which corresponds to P_k and the partition $\{X_2, \ldots, X_m\}$.
- $F'_k = F_k \setminus \{v_k\}$, which fulfills $F'_k \subset F$.

If the root node of $\mathcal{M}$ does not have $\langle x_1, \ldots, x_s \rangle$ as selector variables, this means that $\mathcal{M}$ is also the AMO-MDD representation of all $P_0, \ldots, P_s$,

and hence $F = F_0 = \cdots = F_s$. In this case the theorem trivially holds by induction hypothesis, taking $P = P_0$, $F = F_0$ and $G = G_0$.

From now on, we assume that the root node of $\mathcal{M}$ has $\langle x_1, \ldots, x_s \rangle$ as selector variables. By the definition of Minimal Encoding, we have $F = F'_0 \cup \cdots \cup F'_s \cup \{v_0 \vee \overline{x_0} \vee \overline{v_r}, \ldots, v_s \vee \overline{x_s} \vee \overline{v_r}, v_r\}$.

- Assume that we have an assignment A over the variables of P satisfying G. Then, there is an index $max \in 0..s$ such that $max = 0$ or $x_{max} \in A$, $x_j \notin A$ $\forall_{j \in max+1..s}$, and A satisfies $G_0, \ldots, G_{max}$. We can construct an assignment $B \supset A$ satisfying F as follows: $B = A \cup B_0 \cup \cdots \cup B_s \cup \{v_r\}$, where each $B_k \supset A|_{vars(G_k)}$ is an assignment satisfying F'_k as we show below. Note that several B_k can share auxiliary node variables, because the corresponding F'_k may share such variables (i.e., the child AMO-MDDs of the root of $\mathcal{M}$ may not be disjoint). However, in this proof we show a way to deterministically construct the assignment B, and the same procedure can be applied in the construction of all B_k. This means that B can be consistently constructed, in the sense that a same auxiliary node variable does not have two different values in two different B_k.

  First of all, we have that v_r must be in B to satisfy the clause v_r. By definition of max, A is a model of $G_0, \ldots, G_{max}$, and therefore by induction hypothesis there exist assignments $B_0, \ldots, B_{max}$ satisfying $F'_0 \cup \{v_0\}, \ldots, F'_{max} \cup \{v_{max}\}$, respectively. Then, B also satisfies the formulas $F'_0, \ldots, F'_{max}$ and the clauses $v_0 \vee \overline{x_0} \vee \overline{v_r}, \ldots, v_{max} \vee \overline{x_{max}} \vee \overline{v_r}$. We also know by definition of x_{max} that $\overline{x_j} \in A \subset B$ for all $j \in max + 1..s$, and therefore B satisfies the clauses $v_{max+1} \vee \overline{x_{max+1}} \vee \overline{v_r}, \ldots, v_s \vee \overline{x_s} \vee \overline{v_r}$. Finally, we have to consider the remaining formulas $F'_{max+1}, \ldots, F'_s$. For i in $max+1..s$ we distinguish the following cases:

  $A \models G_k$**:** By induction hypothesis, let B_k be an assignment satisfying $F'_k \cup \{v_k\}$.

  $A \not\models G_k$**:** By induction hypothesis, let B_k be an assignment satisfying F'_k such that $\overline{v_k} \in B_k$.

  Assignments $B_{max+1}, \ldots, B_s$ satisfy $F'_{max+1}, \ldots, F'_s$ respectively, and therefore also does B.

- Assume that we have an assignment A over the variables of P not satisfying G. Then there is an index $max \in 0..s$ such that

$max = 0$ or $x_{max} \in A$, $x_j \notin A \; \forall_{j \in max+1..s}$, and A does not satisfy $G_{max}, \ldots, G_s$.

We will show that there exists an assignment $B \supset A$ that satisfies $F \setminus \{v_r\}$, and any such assignment must assign v_r to *false*. This assignment is $B = A \cup B_0 \cup \cdots \cup B_s \cup \{\overline{v_r}\}$, where each $B_k \supset A|_{vars(G_k)}$ is an assignment satisfying F'_k as we show below. Again, the coherence of the shared variables in different B_k is guaranteed.

Since A is not a model of $G_{max}, \ldots, G_s$, by induction hypothesis let $B_{max}, \ldots, B_n$ be assignments respectively satisfying $F'_{max}, \ldots, F'_s$, and therefore respectively assigning $v_{max}, \ldots, v_s$ to false. We have that $x_{max} \in B$ because either $x_{max} \in A$, or $max = 0$ and $x_0 = \mathit{true}$ by assumption, and we also have that $\overline{v_{max}} \in B$ since $\overline{v_{max}} \in B_{max}$. Therefore, v_r must be false in B in order to satisfy the clause $v_{max} \vee \overline{x_{max}} \vee \overline{v_r}$. In consequence all clauses $v_0 \vee \overline{x_0} \vee \overline{v_r}, \ldots, v_s \vee \overline{x_s} \vee \overline{v_r}$ are also satisfied. Finally, we have two possible cases for each one of the remaining formulas $F'_0, \ldots, F'_{max-1}$:

- $A \models G_k$**:** By induction hypothesis, let B_k be an assignment satisfying $F'_k \cup \{v_k\}$.
- $A \not\models G_k$**:** By induction hypothesis, let B_k be an assignment satisfying F'_k such that $\overline{v_k} \in B_k$.

Such assignments $B_0, \ldots, B_{max-1}$ satisfy $F'_0, \ldots, F'_{max-1}$ respectively and therefore also does B.

□

Corollary 4.3.2. *Let F be a Minimal Encoding for a PB constraint of the form $\sum_{i=1}^{n} q_i x_i \leq K$ with positive coefficients and partition $\{X_1, \ldots, X_m\}$ of its variables. Then F is also an encoding of $\sum_{i=1}^{m} \max_{x_j \in X_i} (q_j \cdot x_j) \leq K$.*

The following corollary states how to use the Minimal Encoding to encode a PB(AMO) constraint.

Corollary 4.3.3. *Let $\mathcal{P}$ be a PB(AMO) constraint of the form $P \wedge M_1 \wedge \cdots \wedge M_m$ with positive coefficients in P. Let F be a Minimal Encoding for P and partition $\{X_1, \ldots, X_m\}$, where $X_i = scope(M_i)$. Let $E(M_1 \wedge \cdots \wedge M_m)$ be an encoding of $M_1 \wedge \cdots \wedge M_m$. Then, $F \wedge E(M_1 \wedge \cdots \wedge M_m)$ is an encoding of $\mathcal{P}$.*

Proof. The corollary follows from Lemma 4.1.2 and Corollary 4.3.2. □

The following theorem states that we can generate an UP-maintaining GAC encoding of a PB(AMO) constraint using the Minimal Encoding.

Theorem 4.3.4. *Let $\mathcal{P}$ be a PB(AMO) constraint of the form $P \wedge M_1 \wedge \cdots \wedge M_m$ with positive coefficients in P. Let F be a Minimal Encoding for P and partition $\{X_1, \ldots, X_m\}$, where $X_i = scope(M_i)$. Let $E(M_1 \wedge \cdots \wedge M_m)$ be a UP-maintaining GAC encoding of $M_1 \wedge \cdots \wedge M_m$. Then, $F \wedge E(M_1 \wedge \cdots \wedge M_m)$ is an UP-maintaining GAC encoding of $\mathcal{P}$.*

Proof. Consider any partial assignment A over the variables of $\mathcal{P}$ that can be extended to a model of $\mathcal{P}$. We need to show that for every variable x of $\mathcal{P}$ such that x is not assigned in A, if $A \cup \{x\}$ cannot be extended to a satisfying assignment of $\mathcal{P}$, then x will be set to false by unit propagating A on $F \wedge E(M_1 \wedge \cdots \wedge M_m)$. Note that, due to the monotonicity of P and of $M_1, \ldots, M_m$, $A \cup \{\overline{x}\}$ can always be extended to a satisfying assignment, so we do not need to consider this case.

First note that, if $A \cup \{x\}$ could be extended to a model of $M_1, \ldots, M_m$, and could also be extended to a model of P then, due to monotonicity, $A \cup \{x\}$ could be extended to a model of $\mathcal{P}$ by setting the remaining variables of $\mathcal{P}$ to false. Therefore, since we assume that $A \cup \{x\}$ cannot be extended to a model of $\mathcal{P}$, we only need to analyse the following two reasons why $A \cup \{x\}$ cannot be extended:

- $A \cup \{x\}$ falsifies some AMO constraint. In such case, the assumption that $E(M_1 \wedge \cdots \wedge M_m)$ UP-maintains GAC will unit-propagate $\overline{x}$.
- $A \cup \{x\}$ can be extended to satisfy $M_1 \wedge \cdots \wedge M_m$ but $A \cup \{x\}$ falsifies P. In this case, the proof is a trivial generalisation of Theorem 23 in [ANO+12].

□

4.4 Other PB($\mathcal{C}$) Constraints

In this section we present other instances of PB modulo $\mathcal{C}$ constraints, and propose how to encode them.

4.4.1 PB(EO) Constraints

In many applications there appear conjunctions of PB constraints with exactly-one (EO) constraints over their variables. A particular case where this happens is when encoding LIA constraints as pseudo-Boolean constraints with a *direct encoding* of the integer variables.

Definition 4.4.1. *By* PB(EO) constraint *we refer to a constraint of the form* $P \wedge E_1 \wedge \cdots \wedge E_m$*, where* P *is a PB constraint,* $E_1, \ldots, E_m$ *are EO constraints, and* $\{scope(E_1), \ldots, scope(E_m)\}$ *is a partition of* $scope(P)$*.*

Since an EO constraint implies an AMO constraint, by Lemma 4.1.3 we can use the presented AMO-MDD based encoding to encode a PB(EO), which by itself is a noticeable improvement with respect to a naive encoding of a PB(EO) constraint.

However, we can do better, by reducing the number of variables of the PB constraint. By subtracting the same integer from all the coefficients of a set of variables holding an EO constraint, as well as to the right hand side of the inequality, we can make some coefficients become zero, and then remove those zero coefficient variables. For example, let P be the PB constraint $2x_1 + 3x_2 + 4x_3 + 3x_4 + 4x_5 + 5x_6 \leq 7$, and suppose we want to encode $P \wedge x_1 + x_2 + x_3 = 1 \wedge x_4 + x_5 + x_6 = 1$. Then we can replace P by $(2-2)x_1 + (3-2)x_2 + (4-2)x_3 + 3x_4 + 4x_5 + 5x_6 \leq 7 - 2$, because exactly one of x_1, x_2 and x_3 will be set to 1 in any assignment satisfying $x_1 + x_2 + x_3 = 1$, and therefore 2 will be subtracted exactly once in the left hand side of the inequality. Similarly, we can subtract 3 to the coefficients of x_4, x_5 and x_6, obtaining $P' : x_2 + 2x_3 + x_5 + 2x_6 \leq 2$. We have that $P \wedge x_1 + x_2 + x_3 = 1 \wedge x_4 + x_5 + x_6 = 1$ is equivalent to $P' \wedge x_1 + x_2 + x_3 = 1 \wedge x_4 + x_5 + x_6 = 1$, with the advantage that P' has a smaller ROBDD representation. Notice that this reduction technique is in fact an abstraction of the non-renewable resource demand reduction that we applied in non-renewable resource constraints in Section 3.2.5. However we can make still more powerful use of the technique, because $x_1 + x_2 + x_3 = 1 \models x_2 + x_3 \leq 1$ and $x_4 + x_5 + x_6 = 1 \models x_5 + x_6 \leq 1$, and therefore by Lemma 4.1.3 we can still use the AMO-MDD based encoding of P'.

4.4.2 PB(EO) and PB(AMO) Constraints with Negative Coefficients

As stated before, most existing encodings of PB constraints to SAT are designed for constraints of the form $\sum_{i=i}^{n} q_i x_i \leq K$, with non-negative q_i, since other cases can be easily transformed to this one. Also the encoding that we have presented in Section 7.1 requires this normalisation. The usual way of getting rid of negative coefficients [ES06] is by using the equality $x = 1 - \overline{x}$, e.g. $-2x_1 + 6x_2 \leq 5 \equiv 2\overline{x_1} + 6x_2 \leq 7$. Then, if we want to encode a constraint of the form $\sum_{i=i}^{n} q_i x_i \geq K$, we can simply replace it by $-\sum_{i=i}^{n} q_i x_i \leq -K$ and get rid of the negative coefficients. However, this rewriting might not be applicable to PB(AMO) constraints. We will illustrate the situation with an example.

Consider the PB(AMO) constraint $P \wedge x_1+x_2+x_3 \leq 1 \wedge x_4+x_5+x_6 \leq 1$, with $P : -x_1-3x_2-4x_3-2x_4-3x_5-5x_6 \leq -6$. If we remove the negative coefficients we obtain $P' : \overline{x_1}+3\overline{x_2}+4\overline{x_3}+2\overline{x_4}+3\overline{x_5}+5\overline{x_6} \leq 12$. The problem now is that $x_1+x_2+x_3 \leq 1$ (similarly $x_4+x_5+x_6 \leq 1$) no longer imposes AMO constraints on the literals of P', and therefore $P' \wedge x_1+x_2+x_3 \leq 1 \wedge x_4+x_5+x_6 \leq 1$ is not a PB(AMO) constraint. We could still use any existing PB encoding to encode P' without taking the AMO constraints into account, but we would not be using the simplification potential of PB(AMO) constraints.

To overcome this weakness, we present another rewriting procedure to get rid of negative coefficients, which does not require to negate the variables of the original PB constraint, and hence still allows us to take into account the AMO constraints. Moreover, this procedure lets us choose the polarity in the PB of any variable by using $x = 1 - \overline{x}$, even if the substitution introduces a negative coefficient, because it will be dealt with in the rewriting. Hence it is possible to make use not only of AMOs between the literals of the PB, but also of AMOs between literals of any polarity.

The first step is, for each AMO constraint of the form $x_{i_1} + \cdots + x_{i_l} \leq 1$, define a fresh variable y_i as $y_i \leftrightarrow \overline{x_{i_1}} \wedge \cdots \wedge \overline{x_{i_l}}$. Then, all the auxiliary y_i variables can be included in the PB constraint with coefficient 0. In the previous example, from:

$$x_1 - 3x_2 - 4x_3 - 2x_4 - 3x_5 - 5x_6 \leq -6 \wedge x_1 + x_2 + x_3 \leq 1 \wedge x_4 + x_5 + x_6 \leq 1$$

$$\begin{aligned} \text{we get :} \qquad & 0y_1 - x_1 - 3x_2 - 4x_3 + 0y_2 - 2x_4 - 3x_5 - 5x_6 \leq -6 \\ & \wedge x_1 + x_2 + x_3 \leq 1 \ \wedge \ x_4 + x_5 + x_6 \leq 1 \\ & \wedge (y_1 \leftrightarrow \overline{x_1} \ \wedge \ \overline{x_2} \wedge \overline{x_3}) \ \wedge \ (y_2 \leftrightarrow \overline{x_4} \wedge \overline{x_5} \wedge \overline{x_6}) \end{aligned}$$

Notice that what we have achieved with this first step is to implicitly introduce an EO constraint for each AMO constraint, because we have that $x_{i_1} + \cdots + x_{i_l} \leq 1 \wedge y_i \leftrightarrow \overline{x_{i_1}} \wedge \cdots \wedge \overline{x_{i_l}} \models y_i + x_{i_1} + \cdots + x_{i_l} = 1$. If we are normalising a PB(EO) constraint instead of a PB(AMO) constraint this first step is not necessary, because the variables of the PB constraint are already partitioned into EO constraints.

After that, for each AMO constraint $x_{i_1} + \cdots + x_{i_l} \leq 1$, choose any integer I_i such that $I_i \geq -q_{i_j}$, for all $1 \leq j \leq l$. For any I_i, it is true that $I_i y_i + I_i x_{i_1} + \cdots + I_i x_{i_l} = I_i$. By adding these equalities to the PB constraint, all the negative coefficients become non-negative, due to the values of I_i that we have chosen. The size of the constraint will not increase if we choose $I = -\min_{i=1}^{n}(q_i)$, since we will be cancelling at least one coefficient of each AMO.

In our example, we get:

$$\begin{aligned}(4+0)y_1 + (4-1)x_1 + (4-3)x_2 + (4-4)x_3 \\ +(5+0)y_2 + (5-2)x_4 + (5-3)x_5 + (5-5)x_6 \leq -6+4+5 \\ \wedge\, x_1 + x_2 + x_3 \leq 1 \;\wedge\; x_4 + x_5 + x_6 \leq 1 \\ \wedge\, (y_1 \leftrightarrow \overline{x_1} \wedge \overline{x_2} \wedge \overline{x_3}) \;\wedge\; (y_2 \leftrightarrow \overline{x_4} \wedge \overline{x_5} \wedge \overline{x_6})\end{aligned}$$

that is:

$$\begin{aligned}4y_1 + 3x_1 + x_2 + 5y_2 + 3x_4 + 2x_5 \leq 3 \\ \wedge\, x_1 + x_2 + x_3 \leq 1 \;\wedge\; x_4 + x_5 + x_6 \leq 1 \\ \wedge\, (y_1 \leftrightarrow \overline{x_1} \wedge \overline{x_2} \wedge \overline{x_3}) \;\wedge\; (y_2 \leftrightarrow \overline{x_4} \wedge \overline{x_5} \wedge \overline{x_6})\end{aligned}$$

Finally, since $y_i + x_{i_1} + \cdots + x_{i_l} = 1 \models y_i + x_{i_1} + \cdots + x_{i_l} \leq 1$, and being F the Minimal Encoding of $4y_1 + 3x_1 + x_2 + 5y_2 + 3x_4 + 2x_5 \leq 3$ with partition $\{\{y_1, x_1, x_2\}, \{y_2, x_4, x_5\}\}$, we can encode the constraint as:

$$\begin{aligned}x_1 + x_2 + x_3 \leq 1 \;\wedge\; x_4 + x_5 + x_6 \leq 1 \\ \wedge\, (y_1 \leftrightarrow \overline{x_1} \wedge \overline{x_2} \wedge \overline{x_3}) \;\wedge\; (y_2 \leftrightarrow \overline{x_4} \wedge \overline{x_5} \wedge \overline{x_6}) \\ \wedge\, F\end{aligned}$$

Note that this procedure to rewrite PB(AMO) constraints is a generalisation of the usual procedure to rewrite PB constraints, since in the case of single-variable AMO constraints of the form $x \leq 1$, instead of defining a fresh variable $y \leftrightarrow \overline{x}$ we can directly use $\overline{x}$ for the transformation and we obtain the same result.

4.4.3 PB(IC) Constraints

[AMES15] introduced an MDD-based encoding for PB constraints with implication chains over their variables. An implication chain (IC) is a constraint of the form $x_1 \leftarrow x_2 \wedge x_2 \leftarrow x_3 \wedge \cdots \wedge x_{n-1} \leftarrow x_n$, denoted $x_1 \Leftarrow x_2 \Leftarrow x_3 \Leftarrow \cdots \Leftarrow x_n$. A PB constraint with a set of implication chains can be seen as another kind of PB($\mathcal{C}$) constraint, that we will denote by PB(IC).

Definition 4.4.2. *By* PB(IC) constraint *we refer to a constraint of the form* $P \wedge IC_1 \wedge \cdots \wedge IC_m$*, where* P *is a PB constraint, and* $IC_1, \ldots, IC_m$ *are implication chains. We assume that every variable in* P *occurs exactly in one* IC_i *(a variable* x *is by itself a single-variable chain).*

As shown in [AMES15], by adding a set of channelling clauses, a PB(IC) encoding can also be used in cases where there is a PB constraint in conjunction with AMO constraints. Although not stated in these terms, they are essentially introducing a different encoding of PB(AMO) constraints. Therefore, we will also evaluate experimentally that encoding in Section 4.5. In that encoding, the AMO constraints have to be encoded with the encoding known as *regular* [AM05], or *ladder* [GN04]. The auxiliary variables introduced by this encoding are involved in an implication chain, and the PB constraint can be reformulated in terms of the new auxiliary variables.

Here we show how to do the translation the other way round, i.e., by adding a set of channelling clauses, a PB(AMO) encoding can be used to encode a PB(IC).

Let the $PB(IC)$ at hand be of the form $q_1x_1 + \cdots + q_nx_n \leq K \wedge IC_1 \wedge \cdots \wedge IC_m$. For each IC_i of the form $x^i_1 \Leftarrow \cdots \Leftarrow x^i_{l_i}$, we add a set of fresh variables $y^i_1, \ldots, y^i_{l_i}$, and the following clauses:

$$y^i_j \leftrightarrow x^i_j \wedge \overline{x^i_{j+1}} \qquad 1 \leq j < l_i \tag{4.3}$$

$$y^i_{l_i} \leftrightarrow x^i_{l_i} \tag{4.4}$$

Finally, we have to encode the PB(AMO) constraint $\sum_{i=1}^{m} \sum_{j=1}^{l_i} {q'}^i_j y^i_j \leq K$, where ${q'}^i_1 = q^i_1$, and ${q'}^i_j = q^i_j + {q'}^i_{j-1}$, for $1 < j \leq l_i$.

For instance, the PB(IC) constraint $2x_1 + 3x_2 + 4x_3 + 6x_4 + 3x_5 \leq 7 \wedge x_1 \Leftarrow x_2 \wedge x_3 \Leftarrow x_4 \Leftarrow x_5$ can be encoded as:

$$\begin{aligned} & x_1 \Leftarrow x_2 \wedge x_3 \Leftarrow x_4 \Leftarrow x_5 \\ \wedge\, & (\, y_1 \leftrightarrow x_1 \wedge \overline{x_2}) \;\wedge\; (y_2 \leftrightarrow x_2) \\ \wedge\, & (y_3 \leftrightarrow x_3 \wedge \overline{x_4}) \wedge\; (y_4 \leftrightarrow x_4 \wedge \overline{x_5}) \\ & \wedge\, (y_5 \leftrightarrow x_5) \wedge\; F \end{aligned}$$

where F is the Minimal Encoding for $2y_1 + 5y_2 + 4y_3 + 10y_4 + 13y_5 \leq 7$ with partition $\{\{y_1, y_2\}, \{y_3, y_4, y_5\}\}$. Note that the two following facts hold:

$$(y_1 \leftrightarrow x_1 \wedge \overline{x_2}) \wedge (y_2 \leftrightarrow x_2) \wedge x_1 \Leftarrow x_2 \models y_1 + y_2 \leq 1$$

$$(y_3 \leftrightarrow x_3 \wedge \overline{x_4}) \wedge (y_4 \leftrightarrow x_4 \wedge \overline{x_5}) \wedge (y_5 \leftrightarrow x_5) \wedge x_3 \Leftarrow x_4 \Leftarrow x_5 \models y_3 + y_4 + y_5 \leq 1$$

Hence, we can use the Minimal Encoding.

4.5 Results

In this section we analyse which is the gain in using the presented MDD-based encoding for PB(AMO) constraints instead of a standard BDD-based encoding, when solving hard combinatorial problems. On one hand, we study the impact on the size of the decision diagrams, and hence on the size of the encodings both in number of variables and clauses. On the other hand, we study the impact on the solving time.

The experiments are run on the MRCPSP and the Resource-Constrained Project Scheduling Problem with Time-Dependent Resource Capacities and Requests (RCPSP/t) [Har13]. We have already seen in Section 2.1.3 that we can encode resource constraints using PBs. We will show in Chapters 5 and 6 that in fact these resource constraints can be formulated as PB(AMO) constraints, the detailed formulations will be shown in those chapters. Here we focus on an experimental comparison between PB and PB(AMO) constraints. We have generated a set of SMT formulas encoding the decision version of the MRCPSP and the RCPSP/t. The PB(AMO) constraints have been encoded in different ways, which we compare:

***BDD*:** All PB constraints are encoded to SAT using the Minimal Encoding of a ROBDD representation, i.e., not taking into account the AMO constraints to simplify the PB encoding. No reduction is done in PB(EO) constraints.

***MDD*:** All PB constraints are encoded to SAT using the Minimal Encoding of an AMO-ROMDD representation. No reduction is done in PB(EO).

***BDDred* and *MDDred*:** The same as *BDD* and *MDD*, respectively, but applying the reduction of Subsection 4.4.1 to all PB(EO) constraints. Only MRCPSP instances have such constraints.

***IC*:** The implication chain based technique to encode PB(AMO) constraints proposed in [AMES15], which we denote by *IC*. The EO reduction is also applied in the instances of MRCPSP.

Using these settings, we have encoded different benchmark instances of the MRCPSP and the RCPSP/t problems from the PSPLIB [KS97] and the MMLIB [VPV14] libraries. We have only used instances for which the optimal makespan is known, hence being able to generate two tight decision cases for every instance, one satisfiable (with an upper bound equal to the optimal makespan) and one unsatisfiable (with an upper bound equal to the optimal makespan minus one). Table 4.1 contains a summary of the different sets

problem	dataset	acts.	name	size	PB(AMO)	PB(EO)
MRCPSP	j30	30	m30sat	545	36132	1090
			m30unsat	545	35037	1090
	MMLIB50	50	m50sat	456	32229	912
			m50unsat	456	31314	912
	MMLIB100	100	m100sat	309	23660	618
			m100unsat	309	23042	618
			total	2620	181414	5240
RCPSP/t	j30	30	t30sat	2826	854900	0
			t30unsat	2826	842328	0
	j120	120	t120sat	2210	1273847	0
			t120unsat	2210	1263954	0
			total	10072	4235029	0

Table 4.1: Generated sets of formulas. The columns contain, in this order: the encoded problem; the original dataset; the number of activities in the instances of this dataset; the name of the generated set, terminated with *sat* if the formulas are satisfiable for the given UB, and with *unsat* otherwise; the number of instances taken (i.e., size of the generated dataset); the total number of PB(AMO) constraints and PB(EO) constraints in the dataset (considering all the formulas).

of formulas created for each setting. The formulas have been generated and solved using the C++ API of Yices 2.4.2 [DdM06a]. All experiments have been run on a 8GB Intel® Xeon® E3-1220v2 machine at 3.10 GHz, with a timeout of 600 seconds in each execution.

We use scatter plots to make pair-wise comparisons of the different settings both for the MRCPSP and the RCPSP/t. In the plots each point corresponds to the value of the same metric in the two compared settings, one in the x axis and the other in the y axis. More precisely, in the left scatter plot of each figure we compare the solving time, where each point corresponds to a different instance, time is in seconds, and we use logarithmic axes. The other plots compare the encoding size in number of variables (middle) and clauses (right), where each point corresponds to a different PB constraint of a different instance, and the values of the axes are in thousands. We also provide a summarised numeric comparison of time and sizes in Tables 4.2 and 4.3.

Figure 4.3 compares the *BDD* and *MDD* settings for all the MRCPSP formulas, and Figure 4.4 shows the same comparison for all the RCPSP/t formulas. In both problems there is a significant reduction in the solving time when using the *MDD* approach, which is up to one order of magnitude in the RCPSP/t, and up to three orders of magnitude in the MRCPSP. For instance, there are formulas which are solved in less than one second with *MDD*, whereas with *BDD* they time out getting lost in the search tree, which contains an order of magnitude more variables. It can be seen that in both problems there is a

	setting	Q1	med	Q3	avg	to
MRCPSP	*BDD*	0.15	0.70	26.24	119.86	473
	BDDred	0.10	0.25	0.97	22.35	44
	MDD	0.06	0.20	0.79	17.71	37
	MDDred	0.05	0.14	0.49	10.27	18
	IC	0.05	0.14	0.52	9.83	17
RCPSP/t	*BDD*	0.24	0.88	3.39	11.92	57
	MDD	0.07	0.33	1.53	4.29	2
	IC	0.09	0.40	1.78	6.24	11

Table 4.2: Solving times for each problem. Columns contain first quartile (Q1), median (med) and third quartile (Q3) of the solving times, average solving time (avg) counting timeouts as 600 seconds, and number of time outs (to).

	variables					clauses				
setting	**Q1**	**med**	**Q3**	**max**	**avg**	**Q1**	**med**	**Q3**	**max**	**avg**
	MRCPSP									
BDD	0.13	0.39	0.92	102.45	1.37	0.26	0.78	1.84	204.90	2.74
BDDred	0.13	0.39	0.90	23.42	0.90	0.26	0.78	1.80	46.84	1.79
MDD	0.02	0.07	0.15	13.12	0.21	0.06	0.29	0.79	52.16	0.89
MDDred	0.02	0.07	0.15	10.87	0.18	0.06	0.29	0.78	26.96	0.76
IC	0.04	0.10	0.20	11.01	0.22	0.10	0.36	0.91	27.19	0.85
	RCPSP/t									
BDD	0.09	0.49	1.62	43.96	1.68	0.18	0.98	3.23	87.91	3.36
MDD	0.00	0.04	0.18	2.34	0.16	0.03	0.33	1.56	40.74	1.80
IC	0.01	0.10	0.31	2.93	0.25	0.08	0.46	1.91	42.51	2.06

Table 4.3: Number of variables and clauses, in thousands, of the encodings of PB constraints. Columns contain first quartile (Q1), median (med), third quartile (Q3), maximum (max) and average (avg) of the sizes of all PB(AMO) and PB(EO) constraints of a problem.

significant reduction in size, which in a large number of constraints is of one order of magnitude. Due to the properties of the encoded problems, there are different clusters of points in the size plots. This is especially noticeable in the plots of the MRCPSP. Each cluster corresponds to PB constraints modelling a particular kind of constraint in a family of instances. There are a subset of constraints in the MRCPSP which have a dramatic size reduction of more than order of magnitude. This subset is in fact the set of PB(EO) constraints, which are the largest PB constraints. In particular, in each of the instances of the MRCPSP dataset that we consider there are two PB(EO) constraints. Since those are the largest PB constraints, we do not observe any impact in the first quartile and median in Table 4.3 when applying the reduction.

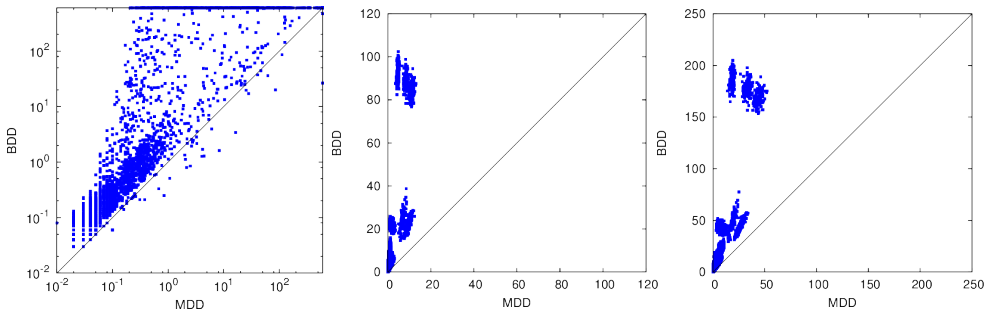


Figure 4.3: Time in seconds (left), number of variables in thousands (middle) and number of clauses in thousands (right) comparison of the *MDD* and *BDD* settings to solve the MRCPSP.

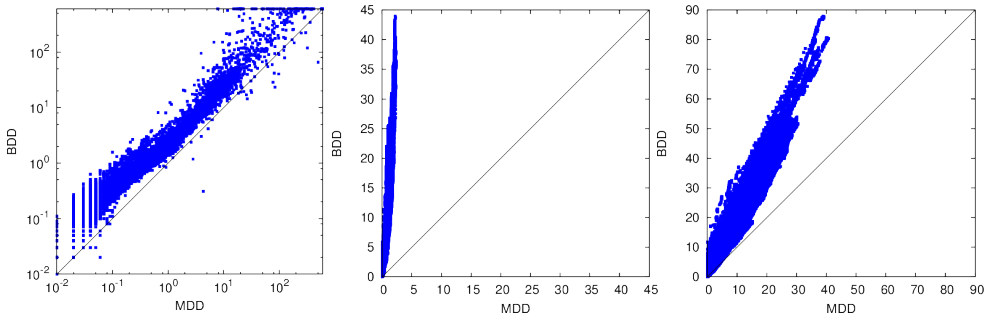


Figure 4.4: Time in seconds (left), number of variables in thousands (middle) and number of clauses in thousands (right) comparison of the *MDD* and *BDD* settings to solve the RCPSP/t.

Figure 4.5 depicts the comparison between *MDD* and *MDDred* in MRCPSP formulas. We observe a further reduction in size, with encodings that are between twice and ten times smaller in most of the cases, and a reduction of the solving time of one order of magnitude.

Figure 4.6 shows the improvement achieved by combining the use of the AMO-MDD based encoding and the reductions by EO constraints (*MDDred*), with respect to the standard BDD-based encoding (*BDD*). In fact, the EO reduction by itself gives a huge improvement, as can be seen in Figure 4.7, which compares the *BDD* and the *BDDred* settings.

When comparing the *BDD* and the *MDD* approaches, the reduction in the number of variables is always strictly higher than the reduction in the number of clauses. This happens because the Minimal Encoding introduces one clause per edge and one variable per node, and the reduction in the number of nodes is higher than the reduction in the number of edges: while the nodes of BDDs

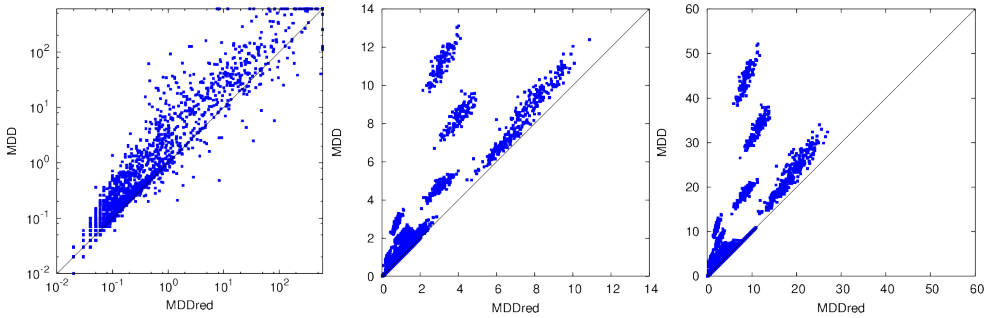


Figure 4.5: Time in seconds (left), number of variables in thousands (middle) and number of clauses in thousands (right) comparison of the *MDDred* and *MDD* settings to solve the MRCPSP.

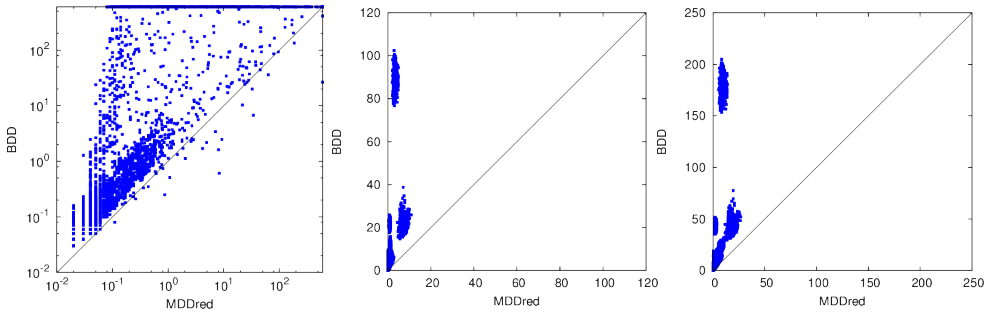


Figure 4.6: Time in seconds (left), number of variables in thousands (middle) and number of clauses in thousands (right) comparison of the *MDDred* and *BDD* settings to solve the MRCPSP.

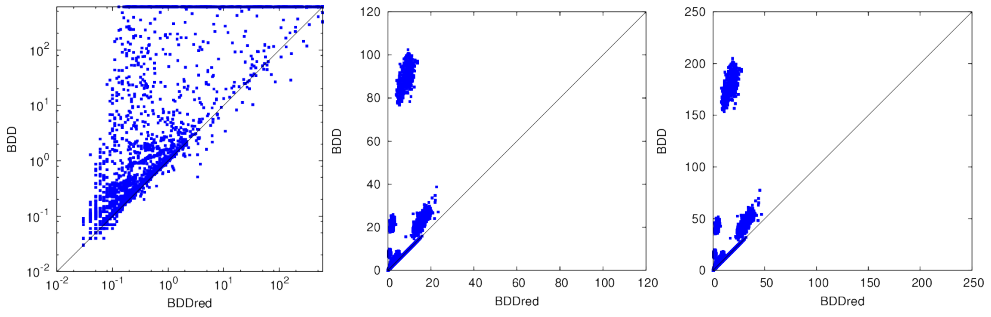


Figure 4.7: Time in seconds (left), number of variables in thousands (middle) and number of clauses in thousands (right) comparison of the *BDDred* and *BDD* settings to solve the MRCPSP.

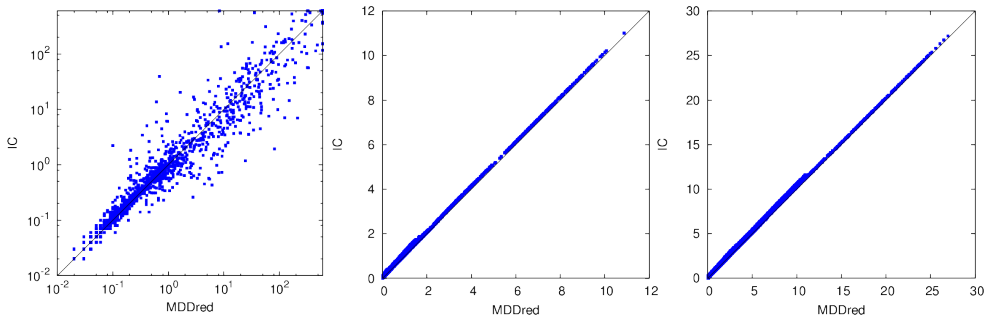


Figure 4.8: Time in seconds (left), number of variables in thousands (middle) and number of clauses in thousands (right) comparison of the *MDDred* and *IC* settings to solve the MRCPSP.

have two outgoing edges, the nodes of AMO-MDDs can have a larger number of edges. Despite this fact, the results show that the number of clauses in the *MDD* approach is noticeably smaller than those in the *BDD* approach.

The reduction in size is directly attributable to the fact that the AMO-MDDs only cover a subset of the possible truth assignments for the variables of the original PB constraints, while the BDDs cover all of them. It is especially noticeable that there are decision diagrams with size 1 in the *MDD* approach, and with size up to 10,000 in the *BDD* approach. The decision diagrams of size 1 correspond to the $\mathcal{T}$-terminal node. This means that there are PB constraints which are trivially true under the assignments that satisfy an AMO constraint on the selector variables of the AMO-MDD nodes.

The depth of the AMO-MDDs is also significantly smaller than the depth of the BDDs. In the considered instances, in most of the cases we are able to include at least three selector variables in each AMO-MDD node, which means that the depth of the AMO-MDDs is at most the depth of the BDDs divided by three. In a large number of cases we are able to get depth reductions of up to seven times, and in few cases the reduction is more than ten times.

Figures 4.8 (MRCPSP) and 4.9 (RCPSP/t) compare the solving times and encoding sizes of the encoding we presented against the *IC* approach. The results suggest that in the MRCPSP both approaches behave similarly, but in the RCPSP/t the AMO-MDD based encoding performs better. The reason may be due to the fact that the RCPSP/t contains bigger AMO constraints, and the *IC* approach needs to explicitly encode the AMOs using a particular encoding that introduces additional Boolean variables and clauses. Size results show this increment, which is only remarkable in the RCPSP/t.

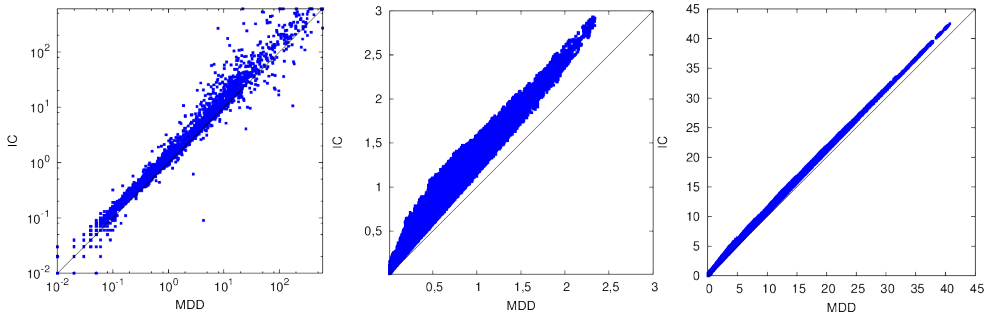


Figure 4.9: Time in seconds (left), number of variables in thousands (middle) and number of clauses in thousands (right) comparison of the *MDD* and *IC* settings to solve the RCPSP/t.

We have run a paired t-test to determine if there is a significant difference in the average run time between the *IC* and *MDD*/*MDDred* approaches, which is not clear apparently in the case of the MRCPSP. We get a p-value of 0.4665 in the MRCPSP, and therefore there is no evidence that one approach is better than the other. Regarding the RCPSP/t, *MDD* is significantly better than *IC*, with a p-value smaller than $2.2e-16$.

4.6 Chapter Summary

With the goal of finding efficient SAT encodings for PB constraints we define PB($\mathcal{C}$) constraints, which are conjunctions of PB constraints and a set of other constraints $\mathcal{C}$ between subsets of the variables of the PB constraint. Then, we propose a framework to encode the PB constraints by assuming that the accompanying constraints are already enforced. We implement this idea in the SAT encoding of PB constraints in presence of AMO relations, by using a specialised and compact type of decision diagrams (AMO-MDD). We provide a UP-maintaining GAC encoding for monotonic decreasing PB(AMO) constraints based on AMO-MDDs. About this encoding, we also state and prove the semantics of the clauses obtained by encoding a PB constraint under the assumption that some AMO constraints hold. Moreover, we show how to convert any PB(AMO) constraint to a monotonic decreasing form, and we tackle other PB($\mathcal{C}$), namely PB(EO) and PB(IC) constraints.

We report a huge impact in the size of the encodings as well as in the solving time when using our encoding approach. The presented techniques are a new and efficient way of handling PB constraints with SAT. Encoding PB($\mathcal{C}$) constraints in the combined way that we propose, may allow reasonably sized SAT encodings that otherwise could be too big for a SAT solver.

Chapter 5

Identifying Collateral AMO Constraints in Scheduling Problems: Application to Efficient SMT Formulations of RCPSP

In the previous chapters of this thesis we have introduced many separate techniques in the line of efficiently solving scheduling problems. In Chapter 3 we proposed a system to solve the MRCPSP, which is a generalisation of the RCPSP whose research interest and applicability is widely acknowledged. We proposed an SMT formulation which was able to outperform the state-of-the-art when solving hard instances. The reason for our success was the combined use of the cheap theory of IDL to specify the precedence relations, and BDD-based SAT encodings of PB constraints to deal with resource constraints. In Chapter 4 we have observed that the existence of AMO constraints in conjunction with PB constraints, i.e. PB(AMO) constraints, can be used to make the translations of the PB constraints into SAT much smaller. Now, in this chapter we show how to integrate all these techniques in order to solve, even more efficiently, RCPSP-based scheduling problems.

The contributions of this chapter are two-fold: on the one hand we show how to efficiently solve the RCPSP using SMT, and on the other hand, we present a template that can be specialised to solve many extensions of the RCPSP, as we will see in Chapter 6. More precisely, we show that in RCPSP-based scheduling problems there naturally appear PB(AMO) constraints. The

PB are the resource constraints, and the accompanying AMO constraints arise from mutual exclusions between the executions of the activities, due to precedence relations. Taking that into account, we propose an SMT formulation where the theory of IDL is used to deal with precedence relations, and SAT encodings of PB(AMO) constraints are used to handle resource constraints using the Time approach. Also, we propose to use a basic and generic ad-hoc exact optimisation algorithm which maintains the learning of the background solver through the search. The result is a state-of-the-art system to solve the RCPSP.

The work presented in this chapter has been partially published in [BCSV17b]. This work is related to the objectives number 1, 3 and 5 of this thesis. The rest of this chapter is organised as follows:

- In Section 5.1 we reproduce the formal RCPSP definition from Section 2.1 to ease the readability of the chapter.
- Section 5.2 explains how to efficiently extract AMO constraints from the precedence graph so that resource constraints can be formulated as PB(AMO) constraints.
- Section 5.3 contains a full SMT formulation of the decision version of the RCPSP.
- Section 5.4 describes a procedure to optimise the makespan of an instance of an RCPSP-based problem.
- Section 5.5 contains an experimental evaluation of our RCPSP formulation.
- In Section 5.6 we summarise the contributions of this chapter.

5.1 RCPSP Definition

The Resource-Constrained Project Scheduling Problem can be formally defined by a tuple (V, p, E, R, B, b) where:

- $V = \{A_0, A_1, \ldots, A_n, A_{n+1}\}$ is a set of activities. Activities A_0 and A_{n+1} are dummy activities introduced by convention, which represent the start and the end of the schedule respectively. They don't consume resources and have duration 0. The set of non-dummy activities is defined by $A = \{A_1, \ldots, A_n\}$.

- $p \in \mathbb{N}^{n+2}$ is a vector of naturals, where p_i is the duration of A_i. For the dummy activities, $p_0 = p_{n+1} = 0$, and $p_i > 0, \forall A_i \in A$.
- E is a set of pairs of activities representing end-start precedence relations. Concretely, $(A_i, A_j) \in E$ iff the execution of activity A_i must precede that of activity A_j, i.e., activity A_j must start after activity A_i has finished. We assume that we are given an activity-on-node *precedence graph* $G = (V, E)$ that contains no cycles, since otherwise the precedence relation is inconsistent. By convention there is a path from A_0 to any other activity, and also a path from any activity to A_{n+1}.
- $R = \{R_1, \ldots, R_v\}$ is a set of renewable resources.
- $B \in \mathbb{N}^v$ is a vector of naturals, where B_k is the available amount of each resource R_k.
- $b \in \mathbb{N}^{(n+2)\times v}$ is a matrix of naturals corresponding to the resource demands of activities, where $b_{i,k}$ represents the amount of resource R_k that activity A_i is using per time step during its execution.

We will denote by $G^* = (V, E^*)$ the extended precedence graph, which has a weight $l_{i,j}$ for each edge $(A_i, A_j) \in E^*$, with $l_{i,j}$ being the critical path from A_i to A_j in G. Having computed G^*, and given a scheduling horizon $H = \{0, \ldots, UB\}$, the different data related with time windows can be obtained as explained in Section 2.1.3: $ES(A_i)$, $LS(A_i)$ and $RTW(A_i)$.

A schedule is a vector of naturals $S = (S_0, S_1, \ldots, S_n, S_{n+1})$ where S_i denotes the start time of activity A_i. We assume that $S_0 = 0$. A solution of the RCPSP is a schedule S of minimal makespan S_{n+1} subject to the precedence and resource constraints. More precisely, the constraints can be formally stated as:

Minimise: $$S_{n+1} \tag{5.1}$$

Subject to:

$$S_j - S_i \geq p_i \qquad \forall (A_i, A_j) \in E \tag{5.2}$$

$$\sum_{A_i \in A} ite\big(S_i \leq t < S_i + p_i;\, b_{i,k};\, 0\big) \leq B_k \qquad \forall R_k \in R, \forall t \in H \tag{5.3}$$

where $ite(c; e_1; e_2)$ is an *if-then-else* expression denoting e_1 if c is true and e_2 otherwise. Precedence constraints (5.2) state that, for any pair $(A_i, A_j) \in E$, activity A_j cannot start until A_i has finished. The renewable resource constraints (5.3) state that the capacities of the renewable resources cannot be exceeded at any time by the demands of the activities running at that time.

5.2 Extracting AMO Constraints from Precedences

As we explained in Chapter 2 in this thesis we will be using formulations of scheduling problems which are based on the Time approach. We have already used this approach in Chapter 3 to formulate renewable resource constraints, either using LIA atoms or encoding PB expressions to SAT. Recall that the basic idea behind this approach is to discretise the time in unit intervals in the scheduling horizon $H = \{0, \ldots, UB\}$, and check that the resource constraints are satisfied at every time. If we want to express a resource constraint as a PB expression, we can be introduce an auxiliary Boolean variable $x_{i,t}$ for each activity A_i and each time $t \in H$, and impose that $x_{i,t}$ is true iff an activity A_i is running at time t. Then, for a resource R_k, the renewable resource constraint of the RCPSP can be stated as:

$$\sum_{A_i \in V} b_{i,k} \cdot x_{i,t} \leq B_k \quad \forall t \in H \tag{5.4}$$

Moreover, this expression can be made more compact if we take into account the time windows of the activities.

Now we show that the variables of the PB renewable resource constraint in RCPSP have AMO relations between them. Therefore Constraint (5.4) can be stated as a PB(AMO) constraint, and hence its translation to SAT can be made much smaller. The key point in efficiently using an encoding of PB(AMO) constraints is being able to identify AMO constraints in the problem at hand. We introduce a technique to identify such AMO constraints in the precedence graph. This method can be used as a basis for formulating resource constraints in many extensions of the RCPSP, as we will see in Chapter 6.

Note that a precedence $(A_i, A_j) \in E$, denoting that A_j cannot start until A_i has finished, introduces an incompatibility between these two activities, i.e. they can never be running at the same time. It is also the case for any pair of activities connected by a path in the precedence graph G, i.e. for any pair of activities with an edge in G^*. Hence the precedence relations ensure that, at any time instant, at most one of the activities of a path of G^* is running. From this information we will infer implicit AMO constraints on variables $x_{i,t}$. For this purpose we will be using path covers:

Definition 5.2.1. *A* path cover *of a graph $G = (V, E)$ is a partition of the vertices in V into a set of sequences $\{P_1, ..., P_l\}$, such that each P_i for $1 \leq i \leq l$ is a path of G. Note that P_i can contain a single vertex. We restrict to vertex-disjoint path covers, i.e. each vertex belongs exactly to one path.*

Definition 5.2.2. *A* minimum path cover *of a graph $G = (V, E)$ is a path cover $\{P_1, \ldots, P_l\}$ such that there does not exist any path cover $\{P'_1, \ldots, P'_{l'}\}$ such that $l' < l$.*

We will compute a path cover $\mathcal{P} = \{P_1, \ldots, P_l\}$ from G^* and then, as we will see in Section 5.3, we will extract an AMO constraint from each P_i. In order to obtain small encodings, a good heuristic is to cover the graph with as few paths of activities as possible. This way, we will minimise the number of parts in which we split the variables of the encoded PB resource constraints. Consequently we will be minimising the number of layers of the AMO-MDD representing the resource constraint, hence obtaining small diagrams. For this reason we will compute $\mathcal{P}$ to be a minimum path cover.

Note the importance of computing the minimum path cover in G^* instead of G. Figure 5.1a depicts a minimum path cover of a precedence graph of a project with 7 activities, and Figure 5.1b shows a minimum path cover of the extended precedence graph. For the sake of simplicity, the extended precedence graph does not show all the additional extended precedences but only $(A_2, A_5) \in E^*$. Thanks to this extended edge, we can find $\mathcal{P} = \{(A_1, A_3, A_4), (A_0, A_2, A_5, A_6)\}$ which only has 2 paths. Otherwise, using the non-extended precedence graph a minimum path cover would contain 3 paths.

Computing a minimum path cover is generally an NP-hard problem, notice that a cover of just one path is a Hamiltonian path. However in directed acyclic graphs (DAG) —like the precedence graph— it can be done in polynomial time. A way to compute the minimum path cover of a DAG is by reducing the problem to finding a maximum cardinality matching of a bipartite graph [NH79].

Definition 5.2.3. *A* maximum cardinality matching *of a graph $G = (V, E)$ is a subset of E such that no two edges share a same node, and that there does not exist any other subset of higher cardinality satisfying this property.*

The reduction from a DAG $G = (V, E)$ can be achieved with the following steps:

1. Create a bipartite graph $G' = (L \cup R, E')$, where all nodes from V are duplicated so that they appear once in L (left partition) and one in R (right partition). For every edge $(V_i, V_j) \in E$, introduce and edge (L_i, R_j) in E'.

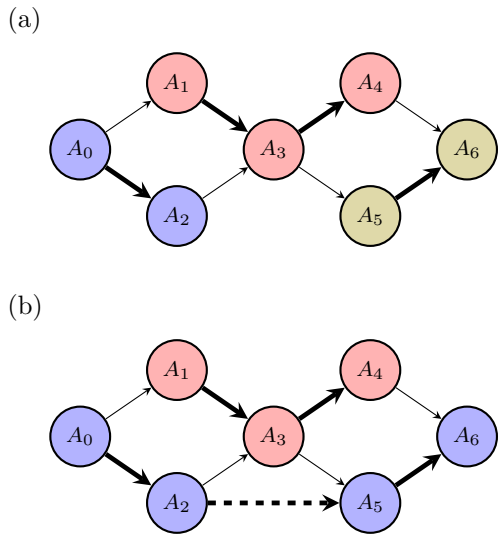


Figure 5.1: Minimum path cover of the precedence graph (a) and the extended precedence graph (b) of a project.

2. Find a maximum cardinality matching.

3. Every selected edge will be part of the path cover. The number of unmatched nodes in either partition L or R is equal to the number of paths of the cover, i.e. every unmatched node in L is the end of a path, and every unmatched node in R is the beginning of a path.

The maximum cardinality matching of a bipartite graph $G = (V, E)$ can be solved in $O(|E|\sqrt{|V|})$ time using the Hopcroft-Karp algorithm [HK73]. Figure 5.2 illustrates the reduction from the extended precedence graph in Figure 5.1b to a maximum cardinality matching.

The minimum path cover yields a way of extracting AMO between variables $x_{i,t}$, since given a time t, all the variables for activities of a same path satisfy an AMO constraint. If the time windows are taken into account, we can focus only on the subgraph containing activities that can be running at a particular time instant t, and probably get smaller covers on G^*. The procedure is the following. For some time instant t:

1. Compute $G^*(t)$ the subgraph from G^* that contains all the nodes of the activities A_i such that $t \in RTW(A_i)$, i.e., $G^*(t) = (V(t), E^*(t))$, where:

$$V(t) = \{A_i \,|\, A_i \in V, s.t. : t \in RTW(A_i)\}$$

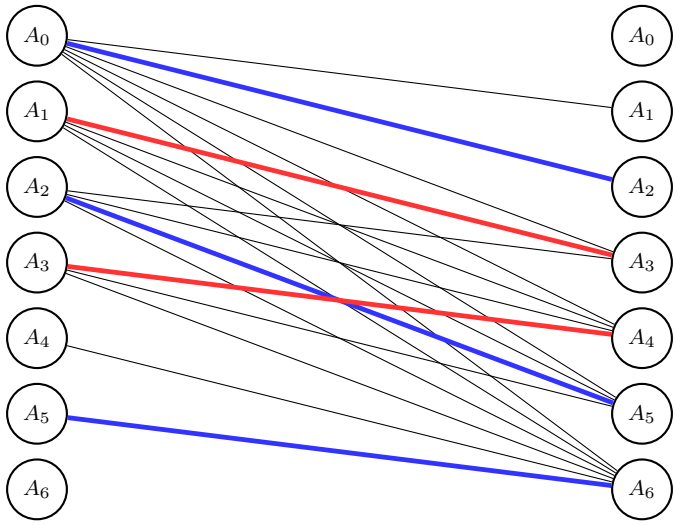


Figure 5.2: Solution of the reduction of the minimum path cover of Figure 5.1b to a maximum cardinality matching in a bipartite graph.

$$E^*(t) = \{(A_i, A_j) \,|\, (A_i, A_j) \in E^*, s.t. : A_i \in V(t) \wedge A_j \in V(t)\}$$

2. Compute a minimum path cover $\mathcal{P}(t) = \{P_1, ..., P_l\}$ of $G^*(t)$.

3. Formulate the renewable resource constraint as a PB(AMO), where the AMOs are extracted from each $P_i \in \mathcal{P}(t)$.

There may be minor differences in these steps depending on the RCPSP variant at hand, and in particular in the ones considered in this paper. The most important is that the set of AMO constraints obtained in the third step has some differences in RCPSP variants. These differences will be described when presenting the formulation of each particular problem.

5.3 SMT Formulation of RCPSP

We provide an SMT formulation for the decision version of the RCPSP, and minimisation of the makespan will be achieved as we explain later in Section 5.4. Therefore, apart from a particular RCPSP instance, we receive as input a precomputed scheduling horizon $H = \{0, \ldots, UB\}$, where UB is the upper bound for the makespan that we want to satisfy. Our SMT formulation of the RCPSP has two sets of variables:

S_i: Integer variables denoting the start time of activity A_i, for all $A_i \in V$.

$x_{i,t}$: Boolean variables which are true iff activity A_i is running at time t, for all $A_i \in A$ and for all $t \in RTW(A_i)$.

The constraints that we add are the following:

$$S_0 = 0 \tag{5.5}$$
$$S_i \geq ES(A_i) \qquad \forall A_i \in \{A_1, \ldots, A_{n+1}\} \tag{5.6}$$
$$S_i \leq LS(A_i) \qquad \forall A_i \in \{A_1, \ldots, A_{n+1}\} \tag{5.7}$$
$$S_j - S_i \geq p_i \qquad \forall (A_i, A_j) \in E \tag{5.8}$$
$$S_j - S_i \geq l_{i,j} \qquad \forall (A_i, A_j, l_{i,j}) \in E^* \tag{5.9}$$
$$x_{i,t} \leftrightarrow S_i \leq t \wedge t < S_i + p_i \qquad \forall A_i \in A,\ \forall t \in RTW(A_i) \tag{5.10}$$
$$\sum_{\substack{A_i \in A,\ s.t.: \\ t \in RTW(A_i)}} b_{i,k} \cdot x_{i,t} \leq B_k \qquad \forall R_k \in R, \forall t \in H \tag{5.11}$$

Constraint (5.5) ensures that the initial dummy activity starts at time 0. Constraints (5.6) and (5.7) bound the start time variables using the time windows. In particular, Constraint (5.7) is the one bounding the makespan, since $LS(A_{n+1}) = UB$. Constraints (5.8) and (5.9) enforce the precedences and the extended precedences respectively. Constraints (5.10) give to variables $x_{i,t}$ the appropriate behaviour according to their semantics. Constraints (5.11) define the renewable resource constraints.

The following constraints are not explicitly imposed, but note that they are logically implied by Constraints (5.8) and (5.10):

$$\sum_{A_j \in P_i} x_{j,t} \leq 1 \qquad \forall t \in H,\ \forall P_i \in \mathcal{P}(t) \tag{5.12}$$

These constraints state that two activities belonging to the same path in $\mathcal{P}(t)$ cannot run at the same time. Therefore Constraints (5.11) and Constraints (5.12) compose a set of PB(AMO) constraints. We will use the Minimal Encoding to encode Constraints (5.11), using the partition of the variables of the PB defined by the AMO constraints (5.12).

5.4 Optimising the Makespan

For RCPSP, as well as the different problems tackled in Chapter 6, we achieve the minimisation of the makespan by making iterative satisfiability checks on the decision version of the problem at hand on a backend SMT solver, while bounding the makespan until the optimum is certified. This procedure is described in Algorithm 8. The most important functions here are:

Algorithm 8 Minimise makespan

Input: Problem instance I
Output: Optimum makespan if satisfiable. Otherwise return unsatisfiable.

```
H ← compute_Horizon(I)
LB ← compute_Lower_Bound(I)
ENC ← smt_encode(I, H)
(SAT, MODEL) ← smt_check(ENC)
if SAT then
  MAKESPAN ← get_makespan(ENC, MODEL)
  UB ← MAKESPAN − 1
else
  return unsat
end if
while SAT and UB ≥ LB do
  ENC ← ENC ∪ {S_{n+1} ≤ UB}
  (SAT, MODEL) ← smt_check(ENC)
  if SAT then
    MAKESPAN ← get_makespan(ENC, MODEL)
    UB ← MAKESPAN − 1
  end if
end while
return MAKESPAN
```

compute_Horizon(I)**:** given an instance I of the problem at hand, it computes a schedule horizon H which is an upper bound of the optimum makespan. For this purpose we use PSGS method, that we described in Section 2.1.3. PSGS is suitable not only for RCPSP, but with few modifications is also suitable for many extensions. In particular, we also use this algorithm for MRCPSP in Section 6.1, RCPSP/t in Section 6.2 and MSPSP in Section 6.4.

compute_Lower_Bound(I) : given an instance I of the problem at hand, it returns a lower bound for the makespan. We will be using the earliest start time of the last dummy activity, i.e. $ES(A_{n+1})$.

smt_encode(I, H)**:** given an instance I of the problem at hand and a scheduling horizon H, it returns an SMT formula representing the problem instance within horizon H.

smt_check(ENC)**:** checks using an SMT solver whether ENC is satisfiable. Returns ($true, MODEL$) if $MODEL$ is a model of ENC, and ($false, \{\}$) otherwise.

get_makespan($MODEL$)**:** retrieves the value of the makespan from a model $MODEL$ of the SMT formula ENC.

It is worth noting that currently there exist many extensions of SAT and SMT which allow to formulate an optimisation problem, and make a single call to a solver that will either provide the optimal solution or prove unsatisfiability. To deal with optimisation in SAT the principal approach is (Weighted) Partial MaxSAT [LM09, ABL13]. In Partial MaxSAT we can distinguish between hard clauses and soft clauses in a formula in CNF, where all the hard clauses must be satisfied as usual, but the soft clauses may be unsatisfied. Then, the problem consists of finding a solution which violates the minimum possible number of clauses. In Weighted Partial MaxSAT there is an associated violation cost for each soft clause. In the SMT setting we can apply Optimisation Modulo Theories (OMT) [ST12], an extension of SMT which allows to directly specify an objective function that the solver will optimise.

Nevertheless, we have conducted some experiments on MRCPSP to evaluate which SMT solver performs better in our system, considering also solvers which offer optimisation capabilities, such as OptiMathSAT [ST15] and Z3 [dMB08]. Yices has given the best performance even though it requires us to implement an ad-hoc optimisation procedure like the one we have described. We have not observed a big reduction on the number of timeouts, but the solving times were generally smaller. Probably using an ad-hoc optimisation algorithm is fast due to the fact that we are optimising a simple objective function regarding expressivity, which can be bounded with just one unit clause of the form $S_{n+1} \leq UB$. Moreover, just adding an upper bound on the makespan enables unit and theory propagation to reduce domains of all start-time variables and time-active variables. It is also very relevant that we are integrating the back-end solver using its API, which allows the solver to keep the learning between different calls.

5.5 Results

In this section we test the performance of our system by solving the most widely used benchmark datasets, that are the *j30*, *j60*, *j90* and *j120* datasets from PSPLib [KS97]. These sets contain instances with 30, 60, 90 and 120 non-dummy activities respectively, and 4 resources. All the sets contain 480 instances except j120, which contains 600 instances.

The two best exact solving approaches that we are aware of are:

- The system presented in [VLS15]. This system implements a search procedure for CP in the *IBM ILOG CPOptimizer* which is specially designed for scheduling problems. The authors name this technique *Failure Directed Search* (FDS), and it resembles the VSIDS heuristic of SAT solvers in the sense that it prioritises the exploration of the configurations that will lead to a conflict. In this paper the authors solved many extensions of the RCPSP, more precisely MRCPSP, RCPSP/max and MRCPSP/max. We contacted the authors, who kindly shared with us their solvers for the different problems. We will refer to this system as *FDS*.

- The system presented in [SFS13]. In that paper the authors develop a *time-table-edge-finding* propagator for cumulative constraints which is integrated into a Lazy Clause Generation solver. The authors report very good results which are available in `https://people.eng.unimelb.edu.au/pstuckey/rcpsp/`. We contacted the authors of that system, and they kindly shared with us some tools and instructions for solvers of other problems different than RCPSP. However they were unable to give us the system for RCPSP that they used in [SFS13], and therefore we have been unable to run their system in our machine for an accurate comparison. We only include the number of timeouts that they report in Table 5.1. They used a machine with a X86-64 architecture running GNU/Linux and a Intel® Core™ i7 CPU processor at 2.8GHz, with the same timeout as us, 600 seconds. We refer to this system as *LCG*.

We have run *FDS* and our system, that we refer to as *SMT*, in a same machine. It is a 8GB Intel® Xeon® E3-1220v2 machine at 3.10 GHz.We use Yices 2.4.2 [DdM06a] as the core SMT solver of our system. Our system is available at the website of the Logic and Programming research group [LAP]. The results are given in Table 5.1.

Looking at the table we can see a clear difference of behaviour between soft instances and hard instances. In all datasets, *FDS* is getting smaller solving times than *SMT* for the first three quartiles, which have very low times never exceeding 2 seconds. This can be explained by the fact that *SMT* is penalised from the construction time of the SMT formula, which contains a lot of clauses related with resource constraints. Also related with the size, the easiest instances may not involve a lot of search and backtracks to be solved, but the number of variables to be assigned is very large in *SMT*. On the other hand, *SMT* is performing better on the hard instances. The average solving time is always strictly smaller in *SMT*, and the number of timeouts is smaller in j60 and j90, and equal in j120.

set	tool	Q1	med.	Q3	avg.	t.o.
j30	FDS	0.01	0.01	0.03	0.93	0
	LCG	—	—	—	—	0
	SMT	0.02	0.03	0.06	0.21	0
j60	FDS	0.01	0.02	0.18	67.44	50
	LCG	—	—	—	—	48
	SMT	0.05	0.12	0.47	62.87	47
j90	FDS	0.02	0.03	0.39	106.95	81
	LCG	—	—	—	—	80
	SMT	0.15	0.36	1.49	104.41	80
j120	FDS	0.05	t.o.	t.o.	322.52	315
	LCG	—	—	—	—	317
	SMT	1.25	t.o.	t.o.	320.23	315

Table 5.1: Solving time results for the instances of RCPSP. All values are in seconds. We give a timeout of 600 seconds to each execution. The first column contains the name of the dataset. Column 2 specifies, for each row, the tool used to solve the instances of the dataset. Columns *Q1*, *med.* and *Q3* specify the first, second, and third quartile of the running times of a dataset (*t.o.* means timeout for that quartile, and dash — means no data available). Column *avg.* contains the average run time, counting timeouts as 600 seconds. The last column contains the number of instances that timed out without having found an optimal solution and proven its optimality.

5.6 Chapter Summary

In this chapter we have provided an SMT formulation of the RCPSP. It uses IDL expressions to deal with precedence constraints, and it uses a Time-indexed approach to express resource constraints. We have also provided a mechanism to detect implicit AMO constraints over the variables of resource constraints, based on the precedence relations. These AMOs let us encode the resource constraints with compact SAT encodings of PB(AMO) constraints. Minimisation of the makespan is efficiently achieved by integrating a back-end SMT solver in an optimisation procedure that increasingly constrains the makespan, and maintains the learning of the solver between calls. Most importantly, the techniques and the formulation presented in this chapter are a basis upon which extensions of the RCPSP can be formulated. Our results show that our system is state-of-the-art in exact solving the hardest instances from PSPLIB of the RCPSP.

Chapter 6

Efficiently Encoding RCPSP-Based Problems to SMT

In this chapter we introduce SMT formulations for four different extensions of the RCPSP which different peculiarities each one. We devote a section to each problem, namely the Multi-mode RCPSP (MRCPSP), the RCPSP with Time-Dependent Resource Capacities and Requests (RCPSP/t), the Multi-mode RCPSP with Minimal and Maximal Time Lags (MRCPSP/max) and the Multi-Skill Project Scheduling Problem (MSPSP). Following the structure of Chapter 5, in each section we first introduce a problem, then we describe it formally, we present our formulations, and finally we provide an experimental section. Since the different problems we consider are extensions of the RCPSP, there are many coincidences in the problem definitions and SMT formulations with those of the RCPSP. However each problem presents completely different variations. Therefore, for the sake of readability we make every section self-contained, and there might be some overlapping with Chapter 5. We will remark the most important peculiarities of each RCPSP extension. Moreover, for some problems we include additional subsections with further contents, as we made other relevant contributions in the problem treated in that section. Regarding experiments, we evaluate the performance of our systems on benchmark instances from the literature, and we compare with the best existing exact solvers, up to our knowledge. Our systems are available at the website of the Logic and Programming research group [LAP]. We run all the solvers on identical machines to get fair comparisons. This has been possible thanks to the authors of the works we compare with, who have kindly

shared their systems and results either in open access platforms or directly with us upon request. Also, they have solved our doubts so that we were able to run their tools using the right configuration, i.e. the one which gave them the best results. We express our sincere gratitude, to all the authors of [Har13, VLS15, SH16, SS16, YFS17].

Since the problems presented in this chapter are extensions of the RCPSP, we will adapt some of the techniques introduced in Chapter 5 to solve them. In particular, there are two main points which we would like to draw attention to:

- We will be using PB(AMO) constraints to formulate resource constraints. In most cases the different problems will be using the pre-computation of $\mathcal{P}(t)$ presented in Section 5.2, i.e. the path cover on a time instant, as a basis to extract implicit AMO constraints. However, each problem may present modifications either in the computation and use of $\mathcal{P}(t)$ or in the PB(AMO) constraint obtained from there.

- We use the generic optimisation procedure described in Section 5.4 to optimise the makespan. We will describe some differences in this procedure in some problems, most notably related with the computation of a scheduling horizon.

The work presented in this chapter has been partially published in two different works. Namely the the work related to MRCPSP and RCPSP/t is partially published in [BCSV17b], and the work related to MRCPSP/max is partially published in [BCSV17a]. This chapter is related to the objectives number 1, 3 and 5 of this thesis. The rest of this chapter is organised as follows.

- In Section 6.1 we provide an SMT formulation for the MRCPSP, which is an improved version of the BDD-based one presented in Chapter 3.

- In Section 6.2 we provide an SMT formulation for the RCPSP/t which, up to our knowledge, is the first exact method that solves this problem.

- In Section 6.3 we provide an SMT formulation for the MRCPSP/max. Also we provide some new benchmark instances, since the existing ones were not very hard regarding resource constraints.

- In Section 6.4 we provide an SMT formulation for the MSPSP, as well as an algorithm that is integrated to the PSGS method to compute a scheduling horizon.

- In Section 6.5 we summarise the contributions of this chapter.

6.1 MRCPSP

In Chapter 3 we already presented two formulations for the MRCPSP which handle resource constraints differently, one using the theory of LIA, and the other one using BDD-based SAT encodings of PB constraints. In this section we present a new formulation for the MRCPSP somewhat similar to the BDD-based one that we introduced in Chapter 3, but with some improvements. In particular we do not introduce variables $x_{i,t}$ but only variables $x_{i,t,o}$, which are now more easily defined. Most importantly, we use SAT encodings of PB(AMO) constraints to deal with resource constraints, instead of SAT encodings of PB constraints. This technique lets us mitigate the effect that the multiple mode choice has over the size of resource PB constraints. This has a huge impact on the solving time performance, as we will see in Section 6.1.3.

We keep using the optimisation procedure from Chapter 3, which is a specialisation for MRCPSP of the algorithm described in Section 5.4. Therefore the formulation we provide is for the decision MRCPSP, given a pre-computed scheduling horizon $H = \{0, \ldots, UB\}$. Recall that the most notorious peculiarities when optimising MRCPSP affect the feasibility determination of the instances, and the computation of the scheduling horizon:

- First of all we solve a relaxed problem in which we only find a schedule of modes which satisfy the non-renewable resource constraints. With this we can certify whether the instance is satisfiable.

- If a schedule of modes has been found in the first step, then we fix these modes to get an instance of the RCPSP, and then we apply the PSGS heuristic to find a scheduling horizon $H = \{0, \ldots, UB\}$.

We include again the problem description for completeness and to ease the readability of the formulation given in Section 6.1.2.

6.1.1 Problem Description

The Multi-mode Resource-Constrained Project Scheduling Problem can be defined by a tuple (V, M, p, E, R, B, b), where:

- $V = \{A_0, A_1, \ldots, A_n, A_{n+1}\}$ is a set of activities. Activities A_0 and A_{n+1} are dummy activities representing, by convention, the start and the end of the schedule, respectively. The set of non-dummy activities is defined by $A = \{A_1, \ldots, A_n\}$.

- $M \in \mathbb{N}^{n+2}$ is a vector of naturals, with M_i being the number of modes that activity A_i can execute in, with $M_0 = M_{n+1} = 1$ and $M_i \geq 1, \forall A_i \in A$.

- p is a vector of vectors of naturals, with $p_{i,o}$ being the duration of activity A_i using mode o, with $1 \leq o \leq M_i$. For the dummy activities, $p_{0,1} = p_{n+1,1} = 0$, and $p_{i,o} > 0, \forall A_i \in A, 1 \leq o \leq M_i$.

- E is a set of pairs of activities representing end-start precedence relations. Concretely, $(A_i, A_j) \in E$ iff the execution of activity A_i must precede that of activity A_j, i.e., activity A_j must start after activity A_i has finished.

  We assume that we are given a precedence activity-on-node graph $G = (V, E)$ that contains no cycles, since otherwise the precedence relation is inconsistent. We assume that E is such that A_0 is a predecessor of all other activities and A_{n+1} is a successor of all other activities.

- $R = \{R_1, \ldots, R_{v-1}, R_v, R_{v+1}, \ldots, R_q\}$ is a set of resources. The first v resources are renewable, and the last $q - v$ resources are non-renewable.

- $B \in \mathbb{N}^q$ is a vector of naturals, with B_k being the available amount of each resource R_k. The first v resource availabilities correspond to the renewable resources, while the last $q - v$ ones correspond to the non-renewable resources.

- b is a three-dimensional matrix of naturals corresponding to the resource demands of activities per mode. $b_{i,k,o}$ represents the amount of resource R_k used during the execution of activity A_i in mode o. Note that $b_{0,k,1} = 0$ and $b_{n+1,k,1} = 0, \forall k \in \{1, \ldots, q\}$.

We will denote by $G^* = (V, E^*)$ the extended precedence graph, which has a weight $l_{i,j}$ for each edge $(A_i, A_j) \in E^*$, being $l_{i,j}$ the critical path from A_i to A_j in G. In the multi-mode case, the weight of an edge (precedence) is the smallest duration among modes of the predecessor activity. Having computed G^*, and given a scheduling horizon $H = \{0, \ldots, UB\}$, the different data related with time windows can be obtained as explained in Section 2.1.3: $ES(A_i)$, $LS(A_i)$ and $RTW(A_i)$.

A schedule is a vector of naturals $S = (S_0, S_1, \ldots, S_n, S_{n+1})$ where S_i denotes the start time of activity A_i. We assume that $S_0 = 0$. A schedule of modes is a vector of naturals $SM = (SM_0, SM_1, \ldots, SM_n, SM_{n+1})$ where SM_i, satisfying $1 \leq SM_i \leq M_i$, denotes the mode of each activity A_i. A solution of the MRCPSP is a schedule of modes SM and a schedule S of minimal

makespan S_{n+1}. It has to respect a correct assignment of execution modes, precedence constraints and renewable and non-renewable resource constraints. More precisely, the constraints can be formally stated as:

$$\text{Minimise:} \qquad S_{n+1} \tag{6.1}$$

Subject to:

$$(SM_i = o) \rightarrow (S_j - S_i \geq p_{i,o}) \qquad \forall (A_i, A_j) \in E, \forall o \in [1, M_i] \tag{6.2}$$

$$1 \leq SM_i \leq M_i \qquad \forall A_i \in V \tag{6.3}$$

$$\sum_{A_i \in A} \sum_{o \in [1, M_i]} ite\big((SM_i = o) \wedge (S_i \leq t < S_i + p_{i,o});\, b_{i,k,o};\, 0\big) \leq B_k \qquad \forall R_k \in \{R_1, \ldots, R_v\}, \forall t \in H \tag{6.4}$$

$$\sum_{A_i \in A} \sum_{o \in [1, M_i]} ite(SM_i = o;\, b_{i,k,o};\, 0) \leq B_k \qquad \forall R_k \in \{R_{v+1}, \ldots, R_q\} \tag{6.5}$$

Precedence constraints (6.2) state that, for any pair $(A_i, A_j) \in E$, activity A_j cannot start until A_i has finished. Constraints (6.3) state that each activity has to run in one of the available modes. The renewable resource constraints (6.4) state that the capacities of the renewable resources cannot be exceeded by the activities running at any particular time. The non-renewable resource constraints (6.5) require that the total demand on a non-renewable resource cannot exceed its availability.

6.1.2 Formulation

Our SMT formulation for the MRCPSP has three sets of variables:

S_i: Integer variables denoting the start time of activity A_i, for all $A_i \in V$.

$sm_{i,o}$: Boolean variables which are true iff activity A_i runs in mode o, for all $A_i \in V$, and for all $o \in [1, M_i]$.

$x_{i,t,o}$: Boolean variables which are true iff activity A_i is running at time t in mode o, for all $A_i \in A$, for all $t \in RTW(A_i)$, and for all $o \in [1, M_i]$.

Then, having precomputed a scheduling horizon $H = \{0, \ldots, UB\}$, the problem is formulated as follows:

$$S_0 = 0 \tag{6.6}$$

$$S_i \geq ES(A_i) \qquad \forall A_i \in \{A_1, \ldots, A_{n+1}\} \tag{6.7}$$

$$S_i \leq LS(A_i) \qquad \forall A_i \in \{A_1, \ldots, A_{n+1}\} \tag{6.8}$$

$$sm_{i,o} \rightarrow S_j - S_i \geq p_{i,o} \qquad \forall (A_i, A_j) \in E, \; \forall o \in [1, M_i] \tag{6.9}$$

$$S_j - S_i \geq l_{i,j} \qquad \forall (A_i, A_j, l_{i,j}) \in E^* \tag{6.10}$$

$$\bigvee_{o \in [1, M_i]} sm_{i,o} \qquad \forall A_i \in V \tag{6.11}$$

$$\overline{sm_{i,o}} \vee \overline{sm_{i,o'}} \qquad \forall A_i \in V, \; \forall o \in [1, M_i - 1], \; \forall o' \in [o+1, M_i] \tag{6.12}$$

$$x_{i,t,o} \leftrightarrow S_i \leq t \wedge t < S_i + p_{i,o} \wedge sm_{i,o} \qquad \forall A_i \in A, \forall o \in [1, M_i], \; \forall t \in RTW(A_i) \tag{6.13}$$

$$\sum_{\substack{A_i \in A, \; s.t.: \\ t \in RTW(A_i)}} \; \sum_{o \in [1, M_i]} b_{i,k,o} \cdot x_{i,t,o} \leq B_k \qquad \forall R_k \in R_1, \ldots, R_v, \; \forall t \in H \tag{6.14}$$

$$\sum_{A_i \in A} \; \sum_{o \in [1, M_i]} b_{i,k,o} \cdot sm_{i,o} \leq B_k \qquad \forall R_k \in R_{v+1}, \ldots, R_q \tag{6.15}$$

Constraint (6.6) ensures that the initial dummy activity starts at time 0. Constraints (6.7) and (6.8) bound the start time variables using the time windows, and by doing that is bounded the makespan S_{n+1}. Constraints (6.9) define the precedence relations, relative to which is the execution mode of the predecessor, and (6.10) define the extended precedences. Constraints (6.11) and (6.12) ensure that each activity runs in exactly one execution mode. Constraints (6.13) give to variables $x_{i,t,o}$ the appropriate behaviour according to their semantics. Constraints (6.14) and (6.15) define the renewable and non-renewable resource constraints respectively.

Similarly as it happens with RCPSP, we can find AMO constraints between the variables of the resource constraints based on precedences. In the

MRCPSP we have a third subindex in variables $x_{i,t,o}$ that is the selected execution mode. However, we can only choose one execution mode per activity, and this introduce additional AMOs. In particular, we have the following implicit constraints in our formulation:

$$\sum_{A_j \in P_i} \sum_{o \in [1,M_j]} x_{j,t,o} \leq 1 \qquad \forall t \in H,\ \forall P_i \in \mathcal{P}(t) \tag{6.16}$$

Intuitively, the implicit Constraints (6.16) state that over all activities connected by a precedence path, at most one of them will be running at a particular time, and in at most one execution mode. These AMO constraints are logically implied by Constraints (6.9), (6.10), (6.12) and (6.13). Therefore, Constraints (6.14) will be encoded using the Minimal Encoding, using the partition of the variables of the PB defined by the implicit AMO constraints (6.16).

Note the great simplification power of using PB(AMO) constraints. For instance, in a project where all activities have three execution modes, the number of variables $x_{i,t,o}$ in a renewable resource constraint would be three times the number that we would get in the single-mode case. However, the number of AMO constraints that we get for each time instant t is exactly the same that we get in the single mode version (cf. Constraints (5.12)), i.e. there is an AMO for each path of the path cover. Therefore, even though the number of variables is increased by a constant factor w.r.t. the single-mode case, the number of layers of the AMO-MDD representation of the constraint will be exactly the same. The number of layers would be otherwise increased if we used a BDD representation of the constraint.

Similarly, Constraints (6.15) compose a PB(EO) constraint together with Constraints (6.11) and (6.12), since the two latter enforce that each activity is running in exactly one mode, i.e. they encode the following constraint:

$$\sum_{o \in [1,M_i]} sm_{i,o} = 1 \qquad \forall A_i \in V \tag{6.17}$$

Therefore we apply the technique to encode PB(EO) constraints explained in Section 4.4.1, i.e. we use the EO reduction to Constraints (6.15) before encoding it with a PB(AMO) encoding. Notice that in practice it is the same as applying the non-renewable demand reduction explained in Section 3.2.5 as a preprocessing step, and then using a PB(AMO) encoding.

6.1.3 Results

We compare our system with the other two best exact systems in the literature (up to our knowledge), which are the following:

set	tool	Q1	med.	Q3	avg.	t.o.
j30	FDS	0.03	0.07	6.4	38.6	15
	LCG	0.11	0.21	0.6	29.6	25
	PB	1.04	2.00	4.4	23.7	14
	PB(AMO)	0.33	0.49	0.8	14.7	9
MMLIB50	FDS	0.04	1.33	t.o.	173.4	140
	LCG	0.59	6.59	338.0	162.4	115
	PB	3.35	7.94	127.3	143.1	105
	PB(AMO)	1.07	1.93	24.2	108.2	88

Table 6.1: Solving time results for the instances of MRCPSP. All values are in seconds. We give a timeout of 600 seconds to each execution. The first column contains the name of the dataset. Column 2 specifies, for each row, the tool used to solve the instances of the dataset. Columns *Q1*, *med.* and *Q3* specify the first, second, and third quartile of the running times of a dataset (*t.o.* means timeout for that quartile). Column *avg.* contains the average run time, counting timeouts as 600 seconds. The last column contains the number of instances that timed out without having found an optimal solution and proven its optimality.

- The system presented in [VLS15], that we also used to solve RCPSP in Chapter 5. We will refer to this system as *FDS*.

- The system presented in [SS16]. It consists of a MiniZinc model for the MRCPSP which uses the global constraint *cumulative* to handle resource constraints, follows a specialised search heuristic, and is solved using a Lazy Clause Generation solver. We refer to this system as *LCG*.

We refer to the results obtained with our SMT formulation as *PB(AMO)*. As stated before the experiments were published in [BCSV17b], and there we also included the results without using MDD-based PB(AMO) encodings for resource constraints but classical BDD-based PB encodings. Also having these results, we can reproduce the decision experiments of Chapter 4 now in the optimisation scenario, and we think that is worthwhile to include them. It is also a way of comparing the improvement of our MRCPSP formulation w.r.t. the one in Chapter 3. We label this setting as *PB*.

We have run all the systems on identical same machines. It is a 8GB Intel® Xeon® E3-1220v2 machine at 3.10 GHz. In all experiments we use Yices 2.4.2 [DdM06a] as the core SMT solver. As we did in Chapter 3, we have tested our proposal on the hard datasets *j30* from PSPLib [KS97] and *MMLIB50* from MMLIB [VPV14]. These results are given in Table 6.1.

PB(AMO) is able to optimally solve within the given timeout a significantly larger amount of instances than the other approaches, especially for MMLIB50, which is the hardest set. In this set, the third quartile of solving times is one order of magnitude smaller than the one of the other approaches. It is also remarkable the improvement w.r.t. using the BDD-based encoding for resource constraints, with many fewer timeouts and smaller solving times. In fact, *PB* is the most penalised approach for the first two quartiles. This can be attributed to the time required to build the BDDs and the big size of the encoding. In Chapter 3 we concluded that our formulation was the best only for hard instances, but thanks to the use of PB(AMO) constraints, now our approach is also very competitive for easy instances.

6.2 RCPSP/t

The *Resource-Constrained Project Scheduling Problem with Time-Dependent Resource Capacities and Requests* (RCPSP/t) is an extension of RCPSP where the demands over resources and the availability of those may vary over time. Although the possibility of having time-dependent demands or availabilities had already been discussed [Har99], it is in [Har13] where the RCPSP/t is introduced, together with a first heuristic solving approach. More precisely, this problem generalises the RCPSP in two aspects. The first one is that the renewable resources can have a different availability at every time instant. The second one is that the resource demands of an activity are not necessarily always the same, but they can take different values at each instant of their execution. This variant of the RCPSP handles periods of interruption of the schedule by setting the capacity of one resource to 0 and making this resource necessary for all the activities. Similarly, it can simulate an interruption of an activity by defining all the resource demands to 0 in some interval of its execution period. These extensions may be hard to handle in some modelling frameworks, but they perfectly fit in a Time formulation as we will see in Section 6.2.2. Similarly to what happened with the MRCPSP, the use of encodings of PB(AMO) constraints will mitigate the size increase of the encodings of resource constraints due to the variability of requests.

6.2.1 Problem Description

The *Resource-Constrained Project Scheduling Problem with Time-Dependent Resource Capacities and Requests* (RCPSP/t) can be defined by a tuple (V, H, p, E, R, B, b) where:

- $V = \{A_0, A_1, \ldots, A_n, A_{n+1}\}$ is a set of activities. Activities A_0 and A_{n+1} are dummy activities representing, by convention, the start and the end of the schedule, respectively. The set of non-dummy activities is defined by $A = \{A_1, \ldots, A_n\}$.

- $H = \{0, \ldots, UB\}$ is the scheduling horizon.

- p is a vector of naturals, with p_i being the duration of activity A_i . For the dummy activities, $p_0 = p_{n+1} = 0$, and $p_i > 0, \forall A_i \in A$.

- E is a set of pairs of activities representing end-start precedence relations. Concretely, $(A_i, A_j) \in E$ iff the execution of activity A_i must precede that of activity A_j, i.e., activity A_j must start after activity A_i has finished.

  Again, we assume that we are given a precedence activity-on-node graph $G = (V, E)$ that contains no cycles. We also assume that A_0 is a predecessor of all other activities and A_{n+1} is a successor of all other activities.

- $R = \{R_1, \ldots, R_v\}$ is a set of renewable resources.

- $B \in \mathbb{N}^{v \times (UB+1)}$ is a matrix of naturals, with $B_{k,t}$ being the available amount of resource R_k at time t. The resources can have a different availability for each time instant in H, and they have an undefined availability in any other time outside the scheduling horizon.

- b is a three-dimensional matrix of naturals corresponding to the resource demands of activities over their execution. $b_{i,k,t}$ represents the amount of resource R_k that activity A_i is occupying during the t-th time instant relative to its start time. Therefore an activity A_i has a vector of demands resource R_k that is $b_{i,k,0}, \ldots, b_{i,k,p_i-1}$.

We will denote by $G^* = (V, E^*)$ the extended precedence graph, which has a weight $l_{i,j}$ for each edge $(A_i, A_j) \in E^*$, with $l_{i,j}$ being the critical path from A_i to A_j in G. Having computed G^*, the different data related with time windows can be obtained as explained in Section 2.1.3: $ES(A_i)$, $LS(A_i)$ and $RTW(A_i)$. For this problem we will also use the start time windows $STW(A_i)$, also explained in Section 2.1.3.

A schedule is a vector of naturals $S = (S_0, S_1, \ldots, S_n, S_{n+1})$ where S_i denotes the start time of activity A_i. We assume that $S_0 = 0$. A solution of the RCPSP/t is a schedule S of minimal makespan S_{n+1}. It has to respect precedence and resource constraints. More precisely, the constraints can be

formally stated as:

$$\text{Minimise:} \qquad S_{n+1} \tag{6.18}$$

Subject to:

$$S_j - S_i \geq p_i \qquad \forall (A_i, A_j) \in E \tag{6.19}$$

$$\sum_{A_i \in A} \sum_{e \in [0, p_i - 1]} ite\big((S_i = t - e);\, b_{i,k,e};\, 0\big) \leq B_k \qquad \forall R_k \in R, \forall t \in H \tag{6.20}$$

Precedence constraints (6.19) state that, for any pair $(A_i, A_j) \in E$, activity A_j cannot start until A_i has finished. The renewable resource constraints (6.20) state that the capacities of the renewable resources cannot be exceeded at any time.

6.2.2 Formulation

In the RCPSP and the MRCPSP formulations, we have been using the Time approach with Boolean variables of the type $x_{i,t}$ which indicate whether an activity A_i is running at time t. However, in the RCPSP/t we need to know not only if an activity is running at a particular time but also for how long it has been running at that time, since the amounts of resource consumptions depend on this. Therefore, just with variables $x_{i,t}$ we cannot formulate the resource constraints of the RCPCP/t. In order to overcome this issue, we use another kind of time-indexed variables with different semantics [Art13]:

$y_{i,t}$**:** 0/1 variable which is true iff activity A_i starts at time t.

As we will see in Constraints (6.27), renewable resource constraints can be stated using variables $y_{i,t}$. When these variables are used the size of the PB constraints increase, because we have to consider all the possible start times which imply that an activity will be running at a certain time. In particular, given a time t, the range of time instants at which an activity A_i can have started so that it is running at time t is $[t - p_i + 1, t]$. However, the advantage of using variables $y_{i,t}$ is that we are able to identify for how long an activity has been running at a particular time instant. As said, this information is required for the RCPSP/t.

The other main difference with respect the RCPSP is that the availability of the resources can be different at any time instant, but this characteristic naturally fits in our time based formulation of the resource constraints, as will be seen in Constraints (6.27). In fact, in the RCPSP/t the scheduling horizon H is part of the definition of an instance, since it contains the time instants for which the availabilities are defined. Nevertheless, we will still compute a tighter UB of the makespan using the PSGS heuristic for the RCPSP, which is also suitable for the RCPSP/t. The are only minor implementation differences with respect to an implementation of PSGS for RCPSP, since it has to be taken into account that the availabilities and requests are not constant.

Our formulation for the RCPSP/t has two sets of variables:

S_i**:** Integer variables denoting the start time of activity A_i, for all $A_i \in V$.

$y_{i,t}$**:** Boolean variables which are true iff activity A_i starts at time t, for all $A_i \in A$, for all $t \in STW(A_i)$.

Then, the problem is formulated as:

$$S_0 = 0 \tag{6.21}$$

$$S_i \geq ES(A_i) \qquad \forall A_i \in \{A_1, \ldots, A_{n+1}\} \tag{6.22}$$

$$S_i \leq LS(A_i) \qquad \forall A_i \in \{A_1, \ldots, A_{n+1}\} \tag{6.23}$$

$$S_j - S_i \geq p_i \qquad \forall (A_i, A_j) \in E \tag{6.24}$$

$$S_j - S_i \geq l_{i,j} \qquad \forall (A_i, A_j, l_{i,j}) \in E^* \tag{6.25}$$

$$y_{i,t} \leftrightarrow S_i = t \qquad \forall A_i \in A,\ \forall t \in STW(A_i) \tag{6.26}$$

$$\sum_{\substack{A_i \in A,\ \text{s.t.}: \\ t \in RTW(A_i)}} \ \sum_{\substack{e \in [0, p_i - 1]\ \text{s.t.}: \\ t - e \in STW(A_i)}} b_{i,k,e} \cdot y_{i,t-e} \leq B_{k,t} \qquad \begin{array}{l} \forall R_k \in R, \\ \forall t \in H \end{array} \tag{6.27}$$

Constraint (6.21) ensures that the initial dummy activity starts at time 0. Constraints (6.22) and (6.23) bound the start time variables using the time windows, and by doing that it is bounded the makespan S_{n+1}. Constraints (6.24) and (6.25) define the precedence and the extended precedence relations respectively. Constraints (6.26) give to variables $y_{i,t}$ the appropriate behaviour according to their semantics. Constraints (6.27) define the renewable resource constraints.

Equally to what happens with variables $x_{i,t}$ for the RCPSP, the precedence relations introduce implicit AMO constraints over variables $y_{i,t}$. Intuitively, if an activity A_i starts at time t, an activity A_j connected by a path in the precedence graph cannot start at time t. Moreover, we know that an activity will only start at one particular time instant, and therefore there is an AMO constraint over all variables $y_{i,t}$ of a same activity A_i. This last AMO is enforced by the channelling between $y_{i,t}$ and the integer start variables S_i, in equation (6.26). Therefore we have the following implicit constraints, which are logically implied by Constraints (6.24) and (6.26):

$$\sum_{A_j \in P_i} \sum_{\substack{e \in [0, p_j - 1], \text{ s.t. :} \\ t - e \in STW(A_j)}} y_{j,t-e} \leq 1 \qquad \forall t \in H,\ \forall P_i \in \mathcal{P}(t) \tag{6.28}$$

The implicit Constraints (6.28) impose that over all activities connected by a path, at most one of them will be running at a particular time, and among all the time instants $t - e$ at which this activity A_i could have started, at most one of $y_{i,t-e}$ can be true. Therefore, Constraints (6.27) will be encoded using the Minimal Encoding, using the partition of the variables of the PB defined by the AMO constraints (6.28).

Similarly to what happens with MRCPSP, the impact on encoding size of having many demand values for a same activity and resource is mitigated by the use of AMO-MDD representations. Note that the number of layers of an AMO-MDD representing a resource constraint (6.27) subject to the AMO constraints (6.28) will be equal to the number of paths in the path cover $\mathcal{P}(t)$. Therefore, the depth of the AMO-MDDs will be the same than the ones of an RCPSP instance of the same characteristics. This would not be the case with classical BDD-based encodings of PB constraints.

6.2.3 Results

To the best of our knowledge, no exact method has been proposed to solve this problem, but only heuristic procedures in [Har13] and [Har15].

We published our RCPSP/t formulation and results in [BCSV17b] together with the ones of the MRCPSP. We reported there a comparison between using AMO-MDD based SAT encoding of PB(AMO) resource constraints to solve the RCPSP/t, against using BDD-based SAT encodings. This is the application to the optimisation scenario of the experiments for the RCPSP/t that we reported in Chapter 4. Now we show these results, comparing these approaches that we label as *PB(AMO)* and *PB*.

We have run our experiments on a 8GB Intel® Xeon® E3-1220v2 machine at 3.10 GHz. In all experiments we use Yices 2.4.2 [DdM06a] as the core

set	tool	Q1	med.	Q3	avg.	t.o.
j30	PB	0.26	1.04	3.1	3.1	0
	PB(AMO)	0.05	0.16	0.4	0.4	0
j120	PB	29.61	t.o.	t.o.	368.1	2085
	PB(AMO)	7.16	56.18	t.o.	253.8	1390

Table 6.2: Solving time results for the instances of RCPSP/t. All values are in seconds. We give a timeout of 600 seconds to each execution. The first column contains the name of the dataset. Column 2 specifies, for each row, the tool used to solve the instances of the dataset. Columns *Q1*, *med.* and *Q3* specify the first, second, and third quartile of the running times of a dataset (*t.o.* means timeout for that quartile). Column *avg.* contains the average run time, counting timeouts as 600 seconds. The last column contains the number of instances that timed out without having found an optimal solution and proven its optimality.

SMT solver. We have tested our proposal on the benchmark datasets that are available in PSPLib [KS97]. These instances were generated in [Har13] and are the only existing ones that we are aware of. They were generated by modifying the *j30* and *j120* instances sets of the RCPSP. The resource demands of the activities among their durations, and the capacities of the resources over time, were modified using different settings. At the end they obtained two datasets:

j30: It contains 2880 instances with 30 non-dummy activities and 4 resources each.

j120: It contains 3600 instances with 120 non-dummy activities and 4 resources each.

The resource availabilities are only specified for the time instants of the scheduling horizon, and therefore any availability for a time outside the horizon is undefined. Some of these instances in the considered datasets do not have feasible schedules within the scheduling horizon specified in the instance files. In these cases, we have considered the instance to be unsatisfiable. The experimental results are contained in Table 6.2.

We are able to close the whole *j30* dataset with both approaches, with an average computation time of 0.4 seconds with *PB(AMO)*. The performance difference in *j120* dataset is very significant, having only 38.6% of the instances timing out with *PB(AMO)* compared to 57.9% with *PB*. With respect to the best results obtained with heuristic methods in [Har13] and [Har15], we have certified the optimality, or infeasibility within the specified scheduling horizon,

of 5090 instances, and improved the best solution of 1763 instances with respect to the heuristic results.

6.3 MRCPSP/max

In this section we tackle the *Multi-mode Resource-Constrained Project Scheduling Problem with Minimal and Maximal Time Lags* (MRCPSP/max). This problem is denoted $MPS|temp|C_{max}$ in [BDM+99] and $m, 1|gpr|C_{max}$ in [HDR99]. It is also known as the Multi-mode RCPSP with Generalised Precedence Relations. This problem is an extension of the MRCPSP which includes the minimal and maximal time lags from RCPSP/max [DRH99]. Like in MRCPSP, the goal is to determine a start time and an execution mode for each activity, in order to obtain a schedule which satisfies all the resource and precedence constraints, and which has minimum makespan. The difference in this problem is that the precedence relations are not the MRCPSP end-start relations, but they define an arbitrary integer time lag between the starts of two activities. A positive time lag is referred to as a minimal time lag, because it can be used to specify minimal time differences between start times. On the other hand, the negative values are referred to as maximal time lags, and are used to specify maximal time differences between starts. The value of the time lag between a pair of activities depends both on the execution mode of the predecessor and the execution mode of the successor.

Most of the recent works on MRCPSP/max have been evaluated using the benchmark datasets created in [Sch98]. However, in [VLS15] it was observed that in those instances the resource constraints are not the hardest component. In this section, apart of proposing an SMT formulation for this problem, we further analyse the reason why the resource constraints seem not to play an important role in those instances, concluding that such constraints are trivially satisfied in many cases. Therefore we have crafted new instance sets, where resource constraints take a more important role. We provide experiments showing that our SMT approach is state of the art among existing exact methods on instances with tight resource constraints.

6.3.1 Problem Description

The MRCPSP/max is defined by a tuple (V, M, p, E, g, R, B, b) where:

- $V = \{A_0, A_1, \ldots, A_n, A_{n+1}\}$ is a set of non-preemptive activities. A_0 and A_{n+1} are dummy activities representing the start and the end of

the schedule, respectively. The set of non-dummy activities is defined as $A = \{A_1, \ldots, A_n\}$.

- $M \in \mathbb{N}^{n+2}$ is a vector of naturals, with M_i being the number of modes in which activity A_i can be executed. $M_0 = M_{n+1} = 1$ and $M_i \geq 1, \forall A_i \in A$.
- p is a vector of vectors of naturals, with $p_{i,o}$ being the duration of activity A_i when is executed in mode o, with $1 \leq o \leq M_i$, $p_{0,1} = p_{n+1,1} = 0$.
- E is a set of pairs of activities which have a time lag defined.
- g is a four dimensional vector of naturals, being $g_{i,j,o,o'}$ the time lag (or generalised precedence relation) from activity A_i in mode o to activity A_j in mode o'. If $g_{i,j,o,o'} >= 0$, it is a minimum time lag, and means that if A_i is running in mode o and A_j is running in mode o', then A_j must start at least $g_{i,j,o,o'}$ units of time after the start of A_i. If $g_{i,j,o,o'} < 0$, it is a maximum time lag, and means that A_i must start at most $|g_{i,j,o,o'}|$ units of time after the start of A_j, if modes o and o' are chosen.
- $R = \{R_1, \ldots, R_{v-1}, R_v, R_{v+1}, \ldots, R_q\}$ is a set of resources. The first v resources are renewable, and the last $q - v$ resources are non-renewable.
- $B \in \mathbb{N}^q$ is a vector of naturals, with B_k being the capacity of resource R_k.
- b is a matrix of naturals corresponding to the resource demands of activities per mode: $b_{i,k,o}$ represents the amount of resource R_k required by activity A_i in mode o, $b_{0,k,1} = 0$ and $b_{n+1,k,1} = 0, \forall k \in \{1, \ldots, q\}$. For renewable resources, this is the required amount per time step, whilst for non-renewable resources, it is the total amount required by the activity during its execution.

We will denote by $G^* = (V, E^*)$ the extended precedence graph, which has a weight $l_{i,j}$ for each edge $(A_i, A_j) \in E^*$, being $l_{i,j}$ the critical path from A_i to A_j in G. In order to compute the critical path, we only consider the edges $(A_i, A_j) \in E$ such that all time lags $g_{i,j,o,'o}$ are positive, and in that case the weight of the edge is the smallest of the time lags. Otherwise the graph could contain negative cycles. Having computed G^*, and given a scheduling horizon $H = \{0, \ldots, UB\}$, the different data related with time windows can be obtained as explained in Section 2.1.3: $ES(A_i)$, $LS(A_i)$ and $RTW(A_i)$.

A schedule is a vector of naturals $S = (S_0, \ldots, S_{n+1})$ where S_i denotes the start time of activity A_i, $S_0 = 0$, and S_{n+1} is the makespan. A schedule of

modes is a vector of naturals $SM = (SM_0, \ldots, SM_{n+1})$ where SM_i, satisfying $1 \leq SM_i \leq M_i$, denotes the mode of each activity A_i. A solution of the MR-CPSP/max problem is a schedule of modes SM and a schedule S of minimal makespan S_{n+1}. It has to respect a correct assignment of execution modes, generalised precedence constraints and renewable and non-renewable resource constraints. More precisely, the constraints can be formally stated as:

Minimise: $$S_{n+1} \tag{6.29}$$

Subject to:

$$\big((SM_i = o) \wedge (SM_j = o')\big) \rightarrow (S_j - S_i \geq g_{i,j,o,o'}) \qquad \forall (A_i, A_j) \in E,\ \forall o \in [1, M_i],\ \forall o' \in [1, M_j] \tag{6.30}$$

$$1 \leq SM_i \leq M_i \qquad \forall A_i \in V \tag{6.31}$$

$$\sum_{A_i \in A} \sum_{o \in [1, M_i]} ite\big((SM_i = o) \wedge (S_i \leq t < S_i + p_{i,o});\ b_{i,k,o};\ 0\big) \leq B_k \qquad \forall R_k \in \{R_1, \ldots, R_v\}, \forall t \in H \tag{6.32}$$

$$\sum_{A_i \in A} \sum_{o \in [1, M_i]} ite(SM_i = o;\ b_{i,k,o};\ 0) \leq B_k \qquad \forall R_k \in \{R_{v+1}, \ldots, R_q\} \tag{6.33}$$

Precedence constraints (6.30) state that, for any pair $(A_i, A_j) \in E$, the time difference between the starts of these activities cannot be smaller than the defined time lag for the selected execution modes. Note that minimal and maximal time lags can be stated equally because of the negative sign of maximal time lags. Constraints (6.31) state that each activity has to run in one of the available modes. The renewable resource constraints (6.32) state that the capacities of the renewable resources cannot be exceeded by the activities running at any particular time. The non-renewable resource constraints (6.33) require that the total demand on a non-renewable resource cannot exceed its availability.

6.3.2 Formulation

Similarly to RCPSP, MRCPSP and RCPSP/t, the SMT formulation that we provide here is for a decisional version of the MRCPSP/max, and optimisation will be achieved by implementing an ad-hoc algorithm for this problem that is described in Section 6.3.3. We also describe there how to pre-compute an initial scheduling horizon.

The formulation that we now introduce is very similar to the one presented for the MRCPSP in Section 6.1. The main observable difference is in Constraints (6.37), that now are generalised precedence relations depending on the execution modes of the two activities involved in the precedence relation. Thanks to the high expressivity of SMT we are able to define this conditioned precedence value with a simple logic expression. However, the generalised precedence relations require us to modify the definition of the precedence path cover $\mathcal{P}(t)$, since now a generalised precedence relation $(A_i, A_j) \in E$ is not an end-start relation, and does not necessarily imply that the two involved activities do not run in parallel. We will be able to detect that two activities A_i, A_j cannot overlap according to the generalised precedence relations if one of the following holds:

- $(A_i, A_j) \in E$, and $g_{i,j,o,o'} \geq p_{i,o}$, for all execution modes o and o'.
- $(A_i, A_j, l_{i,j}) \in E^*$, and $l_{i,j} \geq p_{i,o}$, for all execution modes o.

Hence, $\mathcal{P}(t)$ has to be a path cover of the graph $G'^*(t) = (V(t), E'^*(t))$, where $E'^*(t) \subseteq E^*(t)$ contains only pairs of activities (A_i, A_j) which cannot overlap.

Our formulation for the MRCPSP/max has the same three sets of variables than the MRCPSP:

S_i**:** Integer variables denoting the start time of activity A_i, for all $A_i \in V$.

$sm_{i,o}$**:** Boolean variables which are true iff activity A_i runs in mode o, for all $A_i \in V$, and for all $o \in [1, M_i]$.

$x_{i,t,o}$**:** Boolean variables which are true iff activity A_i is running at time t in mode o, for all $A_i \in A$, for all $t \in RTW(A_i)$, and for all $o \in [1, M_i]$.

Then, the problem is formulated as:

$$S_0 = 0 \tag{6.34}$$

$$S_i \geq ES(A_i) \qquad \forall A_i \in \{A_1, \ldots, A_{n+1}\} \tag{6.35}$$

$$S_i \leq LS(A_i) \qquad \forall A_i \in \{A_1, \ldots, A_{n+1}\} \tag{6.36}$$

$$(sm_{i,o} \wedge sm_{j,o'}) \rightarrow (S_j - S_i \geq g_{i,j,o,o'}) \qquad \forall (A_i, A_j) \in E, \; \forall o \in [1, M_i], \forall o' \in [1, M_j] \tag{6.37}$$

$$S_j - S_i \geq l_{i,j} \qquad \forall (A_i, A_j, l_{i,j}) \in E^* \tag{6.38}$$

$$\bigvee_{o \in [1, M_i]} sm_{i,o} \qquad \forall A_i \in V \tag{6.39}$$

$$\overline{sm_{i,o}} \vee \overline{sm_{i,o'}} \qquad \forall A_i \in V, \; \forall o \in [1, M_i - 1], \; \forall o' \in [o+1, M_i] \tag{6.40}$$

$$x_{i,t,o} \leftrightarrow (S_i \leq t \wedge t < S_i + p_i \wedge sm_{i,o}) \qquad \forall A_i \in A, \forall o \in [1, M_i], \; \forall t \in RTW(A_i) \tag{6.41}$$

$$\sum_{\substack{A_i \in A, \; s.t.: \\ t \in RTW(A_i)}} \; \sum_{o \in [1, M_i]} b_{i,k,o} \cdot x_{i,t,o} \leq B_k \qquad \forall R_k \in R_1, \ldots, R_v, \; \forall t \in H \tag{6.42}$$

$$\sum_{A_i \in A} \; \sum_{o \in [1, M_i]} b_{i,k,o} \cdot sm_{i,o} \leq B_k \qquad \forall R_k \in R_{v+1}, \ldots, R_q \tag{6.43}$$

Constraint (6.34) ensures that the initial dummy activity starts at time 0. Constraints (6.35) and (6.36) bound the start time variables using the time windows, and by doing that the makespan S_{n+1} is bounded. Constraints (6.37) define the generalised precedence relations, relative to which is the execution mode of the two involved activities, and (6.38) define the extended precedences. Constraints (6.39) and (6.40) ensure that each activity runs in exactly one execution mode. Constraints (6.41) give to variables $x_{i,t,o}$ the appropriate behaviour according to their semantics. Constraints (6.42) and (6.43) define the renewable and non-renewable resource constraints respectively.

Similarly to MRCPSP, we have the two following implicit constraints:

$$\sum_{A_j \in P_i} \; \sum_{o \in [1, M_j]} x_{j,t,o} \leq 1 \qquad \forall t \in H, \; \forall P_i \in \mathcal{P}(t) \tag{6.44}$$

$$\sum_{o \in [1, M_i]} sm_{i,o} = 1 \qquad \forall A_i \in V \tag{6.45}$$

Therefore we will encode the resource constraints (6.42) and (6.43) with a PB(AMO) and PB(EO) encoding respectively.

6.3.3 Optimisation

We achieve minimisation of the makespan by applying iterative procedures like the one described in Algorithm 8. However, the generalised precedence relations again make an important difference with respect to the scheduling problems we have considered so far, this time due to the maximal time lags. Let us consider an example time lag $g_{i,j,o,o'} = -3$ for some pair of activities $(A_i, A_j) \in E$ and modes o, o'. This means that, in these modes, *activity A_j must start at least -3 time units after A_i*. This can also be read as *activity A_i cannot start more than 3 time units after A_j*, and this is why this is called a maximal time lag. The PSGS greedy heuristic assumes that there is no restriction on how late an activity can start, but the maximal time lags break this property, and make the PSGS heuristic invalid. Therefore, we will compute a first upper bound of the makespan using a different approach. One possible *UB* could be obtained by trying to make a schedule in which no two activities run at a same time, considering the highest duration for each activity. In order to satisfy minimum time lags in such schedule, after the start of each activity A_i, we can define an inactivity period greater or equal than all the minimal time lags from A_i to any other activity A_j. Even though such a schedule might be inconsistent w.r.t. maximal time lags or non-renewable resource constraints, the optimal makespan will be within the *UB* defined by this schedule. We refer to this bound as the *trivialUB*, and it takes the value:

$$trivialUB = \sum_{A_i \in A} \max_{o \in [1, M_i]} \left(\max \left(p_{i,o}, \max_{\substack{(A_i, A_j) \in E, \\ o' \in [1, M_j]}} g_{i,j,o,o'} \right) \right) \tag{6.46}$$

The *trivialUB* might be very far from the optimum makespan, and this can penalise a Time formulation. Hence, the *trivialUB* is not a good candidate from which to start our search. To overcome this issue, we implement an optimisation procedure inspired in the one presented in [VLS15], which consists of three-steps:

1. **Find a *LB* of the makespan**. We minimise a relaxed version of the MRCPSP/max which does not include renewable resource Constraints (6.42), by following a top-bottom search like the one of Algorithm 8 starting from *trivialUB*. Note that with this relaxation, it is not required to include variables $x_{i,t,o}$ nor Constraints (6.41), and the

number of variables and constraints is independent from the scheduling horizon. Therefore we can start the search from a large UB.

2. **Find a UB of the makespan**. We optimise a single-mode version of the MRCPSP/max, by enforcing the execution modes to be the ones of the optimal solution found in step 1 —note that now Constraints (6.43) can be ignored because the selected modes already satisfy them. We perform a bottom-top search, i.e. we try increasing upper bounds, starting from LB, until a model is found, or *trivialUB* is reached. The solution found in this step (if any) is a valid solution of the original problem, and therefore its makespan is a valid UB. If no solution is found, we use *trivialUB* as UB.

3. **Solve MRCPSP/max**. This last step is only required when $UB > LB$, because otherwise the solution found in the second step is already the optimum of the original problem. If $UB > LB$, we follow the top-bottom search of Algorithm 8.

Steps 1 and 2 are in fact a substitution for the PSGS algorithm, i.e. they let us obtain a first solution whose makespan can be used as UB. There are two main differences with respect to the algorithm from [VLS15]. The first one is that in [VLS15], in step 1 it uses a MIP formulation in which the renewable resource constraints are relaxed using energetic reasoning —instead of completely ignoring them. The second difference is that in [VLS15] built-in optimisation methods are used in all three steps.

6.3.4 New Hard MRCPSP/max Instances

Most of the recent approaches on solving the MRCPSP/max have been evaluated on the datasets generated in [Sch98] and available in the PSPLib [KS97]. In particular there are three datasets, each with 270 problem instances, namely mm30, mm50 and mm100 (instances have 30, 50 and 100 non-dummy activities). The number of execution modes ranges from 3 to 5, and there are 3 renewable and 3 non-renewable resources in each instance. In [VLS15] it was pointed out that, in the previously mentioned instances, resource constraints are not the hardest part of the problem. They used the relaxations on resource constraints mentioned in Section 6.3.3 to find a LB and an UB and it turned out that, in most of the cases, these bounds were equal and therefore an optimal solution was found without the need of encoding the whole original problem.

We have studied why the resources play such a minor role in those datasets. For 1432 out of the total 2430 non-renewable resources constraints, counting

all instances of all datasets, it is trivially true that the demands do not exceed the capacity of the resource, i.e., any assignment of modes satisfies Constraint (6.42) for these resources. We have also checked if the relaxed optimal solutions obtained in step 1 of our optimisation procedure satisfy the renewable resource constraints, which are not enforced in the relaxation of step 1. We have observed that the relaxed optimal solutions found for 593 out of the 810 instances indeed satisfy the renewable resource constraints although they were not encoded, and hence the solutions are also optimal solutions of the original problem. This number may be larger because some instances timed out without having found the optimal solution of the relaxation, and therefore have not been counted.

These characteristics, in addition to the fact that all the mentioned instances have been closed, suggest us that there is a need for new and more challenging datasets, in particular regarding the hardness of resource constraints. The reason why most renewable and non-renewable resources barely constrain the instances is because the capacities are large enough to supply the demands of the activities in a large amount of the possible combinations of mode assignments. For this reason, we propose to use as a basis the same instances, but shrinking the capacities of the resources to amounts which make them non-dummy. For the case of the renewable resources, we have conducted some experiments to see approximately which capacity is needed to make the optimal makespan of an instance increase. We have observed that, given the original demand values and precedence network topologies, for capacities smaller than 30, rarely any instance has an optimal makespan equal to the optimal of the MRCPSP/max without resource constraints. Regarding the non-renewable resources, the original dataset was created with demands ranging from 1 to 10. In the case that all activities required the intermediate amount of 5 units for each non-renewable resource, a capacity of $5n$ would be needed to supply the demands, n being the number of activities of the instance. Considering these facts, we have generated two new versions of each one of the mm30, mm50 and mm100 datasets, namely mm{30,50,100}_1 and mm{30,50,100}_2. They are the result of replacing the resource capacities as stated in Table 6.3. The new mm{30,50,100}_2 datasets are intended to be highly constrained by resources. This is indeed the case, since we have not been able to find any relaxed solution satisfying the renewable resource constraints, and no non-renewable resource constraint is dummy. Sets mm{30,50,100}_1 are a bit easier regarding renewable resources.

Set name	mm30_1	mm50_1	mm100_1
Renewable capacity	[30,39]	[30,39]	[30,39]
Non-renewable capacity	[135,164]	[235,264]	[485,514]

Set name	mm30_2	mm50_2	mm100_2
Renewable capacity	[20,29]	[20,29]	[20,29]
Non-renewable capacity	[135,164]	[235,264]	[485,514]

Table 6.3: Resource capacities of the new datasets. The capacity of each resource of each instance have been generated uniformly and independently at random in the interval indicated in the corresponding cell of the table.

6.3.5 Results

We compare our system, which we refer to as *SMT*, with the other two best exact systems in the literature (up to our knowledge), which are the following:

- The system presented in [VLS15], that we also used to solve RCPSP and MRCPSP. We will refer to this system as *FDS*.

- The system presented in [SH16]. It uses a handler for an extension of the cumulative constraint for the MRCPSP/max integrated into the SCIP optimisation framework. We will refer to this system as *SCIP*.

We have run all three solvers on both the old and the new datasets on identical machines. It is a 8GB Intel® Xeon® E3-1220v2 machine at 3.10 GHz. *SMT* uses Yices 2.4.2 [DdM06a] as the core SMT solver. The results are contained in Table 6.4.

FDS is doubtlessly the best solver for the original datasets, with only 3 timeouts in the hardest dataset (mm100) and average runtimes one order of magnitude lower than the other approaches. This is because its MIP relaxation works extremely well with generalised precedence relations. It must be noted that *SCIP* does not start by solving a relaxed MRCPSP/max with respect to resources, which penalises this approach in these datasets. On the other hand we can see that, in the new datasets, which are more constrained by resources, *SMT* is able to solve more instances than the other approaches in all cases except for the mm100_1 dataset. We remark that 2 out of the only 3 instances that *FDS* solves in this dataset are optimally solved already in the relaxation solving steps.

set	tool	Q1	med.	Q3	avg.	t.o.
mm30	FDS	0.12	0.19	0.32	0.48	0
	SCIP	1.91	2.69	4.68	17.13	1
	SMT	0.52	0.68	0.97	0.87	0
mm50	FDS	0.27	0.49	0.99	1.31	0
	SCIP	6.61	10.89	22.23	78.11	24
	SMT	2.55	3.11	4.16	7.15	0
mm100	FDS	1.25	3.53	10.80	21.69	3
	SCIP	69.91	187.52	t.o.	289.70	90
	SMT	28.25	43.53	527.07	196.05	65
mm30_1	FDS	3.37	8.24	21.76	30.39	1
	SCIP	18.19	49.94	358.43	182.11	49
	SMT	2.64	5.41	12.68	18.55	1
mm50_1	FDS	56.84	271.80	t.o.	324.17	113
	SCIP	t.o.	t.o.	t.o.	496.72	206
	SMT	40.73	249.27	t.o.	312.23	108
mm100_1	FDS	t.o.	t.o.	t.o.	593.74	267
	SCIP	t.o.	t.o.	t.o.	600.00	270
	SMT	t.o.	t.o.	t.o.	600.00	270
mm30_2	FDS	12.43	33.81	130.96	119.77	24
	SCIP	135.56	t.o.	t.o.	420.63	155
	SMT	7.01	19.01	68.71	78.61	12
mm50_2	FDS	t.o.	t.o.	t.o.	517.08	216
	SCIP	t.o.	t.o.	t.o.	584.68	257
	SMT	473.73	t.o.	t.o.	491.52	192
mm100_2	FDS	t.o.	t.o.	t.o.	600.00	270
	SCIP	t.o.	t.o.	t.o.	600.00	270
	SMT	t.o.	t.o.	t.o.	600.00	270

Table 6.4: Solving time results for the instances of MRCPSP/max. All values are in seconds. We give a timeout of 600 seconds to each execution. The first column contains the name of the dataset. Column 2 specifies, for each row, the tool used to solve the instances of the dataset. Columns *Q1*, *med.* and *Q3* specify the first, second, and third quartile of the running times of a dataset (*t.o.* means timeout for that quartile). Column *avg.* contains the average run time, counting timeouts as 600 seconds. The last column contains the number of instances that timed out without having found an optimal solution and proven its optimality.

6.4 MSPSP

The different scheduling problems that we have tackled so far in this chapter are generalisations of the RCPSP, and therefore an RCPSP instance can be stated as an instance of the more generic problem. In this section we tackle the *Multi-Skill Project Scheduling Problem* (MSPSP), which also generalises the RCPSP but presents an essential redefinition of the concept of resource constraints. Like the RCPSP, the MSPSP consists of a project with a set of non-preemptive activities to schedule, there are end-start precedence relations to respect, and the makespan has to be minimised. However, in the MSPSP the activities do not directly ask for resources but they ask for skills. These skills are supplied by renewable resources, and every resource is specialised to master a subset of the skills. A clear example is that the resources are workers. A worker can master many skills, and he/she can perform a different skill on each activity. The resource constraints state that one resource (worker) can only work at one skill of one activity at a time, and that a resource can only supply skills that it masters. The resources are unary, i.e. they can only supply one unit of skill at a time, but the activities may require many units of each skill. Also, the set of resources that an activity is using cannot change at any moment of execution, i.e. the resource usage of the activities is also non-preemptive.

The MSPSP is also known as *Project Scheduling with Flexible Resources* [CLSdG12]. It was argued in [BM08] that an instance of the MSPSP can also be stated as an instance of the MRCPSP, by simulating the different combinations of resource assignments to an activity as execution modes. However, the resulting number of execution modes can become very large due to the combinatorial explosion. Therefore, the MSPSP is usually tackled with specific techniques for this problem. Some MILP models have been studied to solve this problem [BM08, Cas12, CLSdG12], with the Time-based ones being especially efficient. In the latter it was proposed to use a set of cumulative cuts on the MILP model, which are similar to the renewable resource constraints of the RCPSP. This gives a very important speedup of the solving process. [YFS17] proposed a CP model together with a search strategy, which was then solved using a LCG solver. There, the authors also take advantage of the cumulative cuts, which in the CP setting can be seen as implied or redundant constraints.

In this section we propose an SMT formulation for the MSPSP. We also take into account the cumulative implied constraints from [Cas12], that in our formulation can be expressed as PB(AMO) constraints, and therefore we can compactly encode them to SAT. Our results show that the presented formulations are state-of-the-art in exact solving for the MSPSP.

6.4.1 Problem Description

The *Multi-Skill Project Scheduling Problem* (MSPSP) is defined by a tuple (V, p, E, R, L, m, b) where:

- $V = \{A_0, A_1, \ldots, A_n, A_{n+1}\}$ is a set of activities. Activities A_0 and A_{n+1} are dummy activities representing, by convention, the start and the end of the schedule, respectively. The set of non-dummy activities is defined by $A = \{A_1, \ldots, A_n\}$.

- p is a vector of naturals, with p_i being the duration of activity A_i . For the dummy activities, $p_0 = p_{n+1} = 0$, and $p_i > 0, \forall A_i \in A$.

- E is a set of pairs of activities representing end-start precedence relations. Concretely, $(A_i, A_j) \in E$ iff the execution of activity A_i must precede that of activity A_j, i.e., activity A_j must start after activity A_i has finished.

  Again we assume that we are given a precedence activity-on-node graph $G = (V, E)$ that contains no cycles, that A_0 is a predecessor of all other activities and A_{n+1} is a successor of all other activities.

- $R = \{R_1, \ldots, R_v\}$ is a set of unary renewable resources.

- $L = \{L_1, \ldots, L_s\}$ is a set of skills.

- $m \in \mathbb{B}^{v \times s}$ is a matrix of Booleans, with $m_{k,l}$ being true iff resource R_k masters skill L_l.

- $b \in \mathbb{N}^{(n+2) \times s}$ is a matrix of naturals, where $b_{i,l}$ represents the number of resources mastering skill L_l that activity A_i requires during its execution. The start and end dummy activities do not require skills, i.e. $b_{0,l}$ and $b_{n+1,l}$ are 0 for any skill L_l.

We will denote by $G^* = (V, E^*)$ the extended precedence graph, which has a weight $l_{i,j}$ for each edge $(A_i, A_j) \in E^*$, being $l_{i,j}$ the critical path from A_i to A_j in G. Having computed G^*, and given a scheduling horizon $H = \{0, \ldots, UB\}$, the different data related with time windows can be obtained as explained in Section 2.1.3: $ES(A_i)$, $LS(A_i)$ and $RTW(A_i)$.

A schedule is a vector of naturals $S = (S_0, S_1, \ldots, S_n, S_{n+1})$ where S_i denotes the start time of activity A_i. A resource assignment RA is a matrix of three dimensions of Booleans, where $RA_{i,k,l}$ is true iff activity A_i uses resource R_k to perform skill L_l. A solution of the MSPSP is a schedule S of minimal

makespan S_{n+1} and a resource assignment RA subject to the following constraints:

Minimise: $$S_{n+1} \tag{6.47}$$

Subject to:

$$S_j - S_i \geq p_i \qquad \forall (A_i, A_j) \in E \tag{6.48}$$

$$\overline{m_{k,l}} \rightarrow \overline{RA_{i,k,l}} \qquad \forall A_i \in A,\, \forall R_k \in R,\, \forall L_l \in L \tag{6.49}$$

$$\sum_{R_k \in R} RA_{i,k,l} = b_{i,l} \qquad \forall A_i \in A,\, \forall L_l \in L \tag{6.50}$$

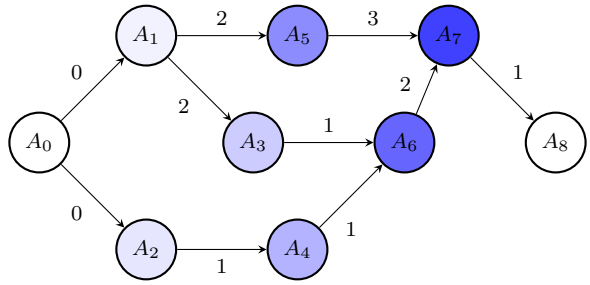
$$\sum_{L_l \in L} \sum_{A_i \in A} ite\big((S_i \leq t < S_i + p_i);\, RA_{i,k,l};\, 0\big) \leq 1 \qquad \forall R_k \in R, \forall t \in H \tag{6.51}$$

Precedence constraints (6.48) state that, for any pair $(A_i, A_j) \in E$, activity A_j cannot start until A_i has finished. Constraints (6.49) state that a resource cannot perform a skill that it does not master. Constraints (6.50) state that each activity must have the required number of resources covering each one of the skills. Constraints (6.51) state that a resource can only work at one skill of one activity at a time.

Figure 6.1 shows an example MSPSP instance with 7 non-dummy activities, 7 resources and three skills. An optimal solution of this instance is shown in Figure 6.2.

6.4.2 Formulation

As stated in the introduction of this section, we will use a set of cumulative implied constraints proposed in [Cas12]. These constraints state that the number of activities running at a particular time cannot be greater than the number of resources required to supply their skills requirements. These implied constraints can be stated for any subset of the skills. Considering the example instance of Figure 6.1, the number of resources mastering skill L_1 is 3. Therefore, any subset of activities requiring more than 3 units of skill L_1 cannot run in parallel at any time. Using an RCPSP-like Time formulation,



	A_0	A_1	A_2	A_3	A_4	A_5	A_6	A_7	A_8
L_1	0	2	1	2	0	1	0	1	0
L_2	0	2	0	1	2	1	0	1	0
L_3	0	2	1	1	1	0	1	1	0

	R_1	R_2	R_3	R_4	R_5	R_6	R_7
L_1	0	0	0	1	0	1	1
L_2	1	1	1	0	1	1	1
L_3	0	0	0	0	1	1	1

Figure 6.1: Example MSPSP instance with 7 non-dummy activities, 3 skills and 7 resources. At the top: precedence graph, where the edge weight is the duration of the predecessor activity. In the middle, transposed demand matrix b. At the bottom, transposed mastery matrix m.

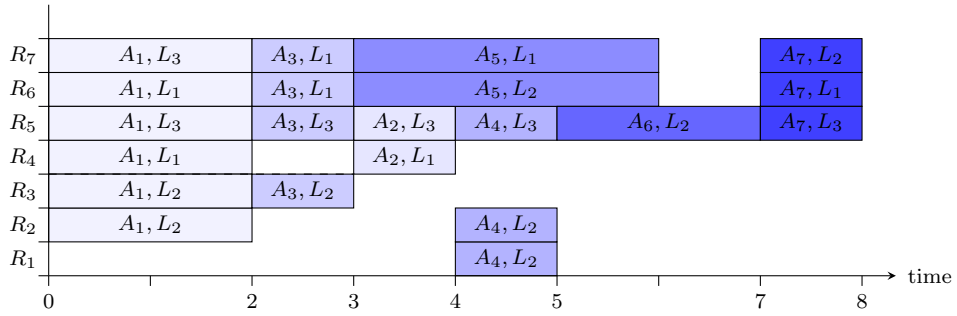


Figure 6.2: Optimal solution of the instance of Figure 6.1, with schedule $S = (0, 0, 3, 2, 4, 3, 5, 7, 8)$. Each box shows the period in which a resource is supplying a skill for an activity.

with activity $x_{i,t}$ denoting whether activity A_i runs at time t, we could state that:

$$\sum_{A_i \in A} b_{i,1} \cdot x_{i,t} \leq 3 \qquad \forall t \in H$$

Note that we could apply the same reasoning not only to L_1, but also to any subset of L. For instance, we know that the number of resources that master one of $\{L_1, L_2\}$ is 7. Hence, we can state that:

$$\sum_{A_i \in A} \left(x_{i,t} \cdot \sum_{L_l \in \{L_1, L_2\}} b_{i,l} \right) \leq 7 \qquad \forall t \in H$$

Let us denote by $\mathcal{PW}(L)$ the power set of L, and by $RL(L')$ the set of resources that master at least one of the skills in L', with $L' \in \mathcal{PW}(L)$. In general, the following must hold:

$$\sum_{A_i \in A} \left(x_{i,t} \cdot \sum_{L_l \in L'} b_{i,l} \right) \leq |RL(L')| \qquad \forall L' \in \mathcal{PW}(L),\ \forall t \in H$$

The number of these implied constraints can become very large in cases where there is a large number of skills. However, in [Cas12] it was observed that most of these constraints become dominated by others, and can be removed. In particular, they were able to remove between 60% and 90% of the constraints in most instances. We will illustrate this dominance analysis continuing with our example of Figure 6.1. The number of resources mastering some skill in L is 7, exactly the same that the resources mastering some of $\{L_1, L_2\}$. Therefore, the following implied constraint is stronger that the one we obtained with $\{L_1, L_2\}$:

$$\sum_{A_i \in A} \left(x_{i,t} \cdot \sum_{L_l \in L} b_{i,l} \right) \leq 7 \qquad \forall t \in H$$

We will say that L' *is dominated by* L'', iff $L' \subset L'' \wedge |RL(L')| = |RL(L'')|$. Then, we denote by $\mathcal{PWR}(L)$ the subset of $\mathcal{PW}(L)$ that contains all subsets of L not dominated by any other set of skills in $\mathcal{PW}(L)$. We will only encode implied constraints for sets of skills in $\mathcal{PWR}(L)$.

Now we present a Time-based SMT formulation for the MSPSP. Again, it is a decision version which will be optimised using Algorithm 8 described in Section 5.4. In order to compute a first solution to get a good UB, we use an adaptation of the PSGS that we describe in Section 6.4.3.

We will use many sets of variables with combined semantics, with the aim of reducing the number of clauses and the size of those. We define the following two additional functions that we use to reduce the number of variables and clauses of the formulation.

$R(A_i)$**:** This is the subset of resources that master at least one of the skills demanded by activity A_i.

$L(A_i)$**:** This is the subset of skills that activity A_i demand, i.e. the skills L_l such that $b_{i,l} > 0$.

The sets of variables that we introduce are the following:

S_i**:** Integer variables denoting the start time of activity A_i, for all $A_i \in V$.

$x_{i,t}$**:** Boolean variables which are true iff activity A_i is running at time t, for all $A_i \in A$, and for all $t \in RTW(A)$.

$ar_{i,k}$**:** Boolean variables which are true iff activity A_i uses resource R_k, for all $A_i \in A$, and for all $R_k \in R(A_i)$.

$ars_{i,k,l}$**:** Boolean variables which are true iff activity A_i uses resource R_k for skill L_l, for all $A_i \in A$, for all $R_k \in R(A_i)$, and for all $L_l \in L(A_i)$ such that $m_{k,l}$, i.e. such that resource R_k masters skill L_l.

$art_{i,k,t}$**:** Boolean variables which are true iff activity A_i uses resource R_k at time t, for all $A_i \in A$, for all $R_k \in R(A_i)$, and for all $t \in RTW(A_i)$.

Using these variables we can define our SMT formulation:

$$S_0 = 0 \tag{6.52}$$

$$S_i \geq ES(A_i) \qquad \forall A_i \in \{A_1, \ldots, A_{n+1}\} \tag{6.53}$$

$$S_i \leq LS(A_i) \qquad \forall A_i \in \{A_1, \ldots, A_{n+1}\} \tag{6.54}$$

$$S_j - S_i \geq p_i \qquad \forall (A_i, A_j) \in E \tag{6.55}$$

$$S_j - S_i \geq l_{i,j} \qquad \forall (A_i, A_j, l_{i,j}) \in E^* \tag{6.56}$$

$$ar_{i,k} \leftrightarrow \bigvee_{\substack{L_l \in L(A_i),\\ s.t.:\ m_{k,l}}} ars_{i,k,l} \qquad \forall A_i \in A,\ \forall R_k \in R(A_i) \tag{6.57}$$

$$x_{i,t} \leftrightarrow S_i \leq t \wedge t < S_i + p_i \qquad \forall A_i \in A,\ \forall t \in RTW(A_i) \tag{6.58}$$

$$art_{i,k,t} \leftrightarrow ar_{i,k} \wedge x_{i,t} \qquad \forall A_i \in A,\ \forall R_k \in R(A_i),\ \forall t \in RTW(A_i) \tag{6.59}$$

$$\sum_{\substack{R_k \in R,\\ s.t.:\ m_{k,l}}} ars_{i,k,l} = b_{i,l} \qquad \forall A_i \in A,\ \forall L_l \in L(A_i) \tag{6.60}$$

$$\sum_{\substack{L_l \in L(A_i),\\ s.t.:\ m_{k,l}}} ars_{i,k,l} \leq 1 \qquad \forall A_i \in A,\ \forall R_k \in R(A_i) \tag{6.61}$$

$$\sum_{\substack{A_i \in A,\ s.t.:\\ t \in RTW(A_i),\\ R_k \in R(A_i)}} art_{i,k,t} \leq 1 \qquad \forall R_k \in R,\ \forall t \in H \tag{6.62}$$

$$\sum_{\substack{A_i \in A,\ s.t.:\\ t \in RTW(A_i)}} \left(x_{i,t} \cdot \sum_{L_l \in L'} b_{i,l} \right) \leq |RL(L')| \qquad \forall L' \in \mathcal{PWR}(L), \forall t \in H \tag{6.63}$$

Constraint (6.52) ensures that the initial dummy activity starts at time 0. Constraints (6.53) and (6.54) bound the start time variables using the time windows, and by doing that it is bounded the makespan S_{n+1}. Constraints (6.55) and (6.56) define the precedences and the extended precedences respectively. Constraints (6.57), (6.58) and (6.59) are channelling constraints defining the semantics of variables $ar_{i,k}$, $x_{i,t}$ and $art_{i,k,t}$ respectively. Constraints (6.60)

ensure that each activity has enough resources to cover each one of its skills. Note that it is implicitly forbidden that a resource R_k performs a skill L_l that it does not master, since the corresponding variable $ars_{i,k,l}$ is not introduced. These are cardinality constraints that we encode to SAT using the sorting network based encoding from [ANORC13]. Constraints (6.61) state that a resource can contribute with at most one skill in one activity. Constraints (6.62) ensure that a resource does not contribute to more than one activity at a time. Both Constraints (6.61) and (6.62) are AMO constraints, that we encode into SAT using pairwise mutual exclusions between variables, i.e. clauses of the form $\overline{x} \vee \overline{y}$ for any pair of variables in the AMO constraint. Constraints (6.63) are the cumulative implied constraints, taking into account time windows.

Note that Constraints (6.63) are PB, and they contain variables $x_{i,t}$ with the same semantics as resource constraints in RCPSP. Therefore there are implicit AMO constraints over variables introduced by precedences. The following implicit AMO constraints are logical consequence of Constraints (6.55) , (6.56) and (6.58):

$$\sum_{A_j \in P_i} x_{j,t} \leq 1 \qquad \forall t \in H,\ \forall P_i \in \mathcal{P}(t) \tag{6.64}$$

We will use the Minimal Encoding for PB(AMO) constraints to encode Constraints (6.63) with the variable partitions given by AMO Constraints (6.64).

6.4.3 UB Computation

In order to compute an *UB* for the makespan we use the PSGS method, tuned to also deal with resource assignments. Recall that PSGS, described in Section 2.1.3, greedily constructs a schedule by trying to set the start time of as much activities as possible at time t, respecting precedence and resource constraints. When no more activities can be placed at time t, it moves to a greater time instant and continues the procedure. Therefore, one of the sub-problems contained in this procedure is to determine whether an activity can start at a given time t, taking into account that a (possibly empty) set of activities have already been scheduled and are running at time t, and therefore some amount of the resources are being used. In the RCPSP it is trivially checkable if there is enough free capacity of the resources. However this is not straightforward in the MSPSP, since we have to make sure that the resource assignment of an activity remains the same during its execution.

For this purpose we have implemented Algorithm 9, which receives: a vector of skill requirements of an activity $\beta = b_1, \ldots, b_s$; a set of free resources $FR \subseteq R$; the mastery matrix m; an input/output set U of pairs of the form

Algorithm 9 FindResAssignment

Input: vector $\beta : b_1, \ldots, b_s$, set FR of free resources, mastery matrix m of size $v \times s$, input/output set of unfeasible U

Output: returns $(\alpha, true)$ if an assignment α can be constructed, $(\{\}, false)$ otherwise

1: **if** $\sum_{l=1}^{s} b_l = 0$ **then return** $(\{\}, true)$ **end if**
2: **if** $\sum_{l=1}^{s} b_l \leq |FR|$ **then**
3: $R_k \leftarrow getFirst(FR)$
4: $FR' \leftarrow FR \setminus \{R_k\}$
5: **for all** $l \in 1..s$ **do**
6: **if** $m_{k,l}$ **and** $b_l > 0$ **then**
7: $\beta' \leftarrow decreaseBy1(\beta, l)$
8: **if** $(\beta', FR') \notin U$ **then**
9: $(\alpha, feas) \leftarrow$ **FindResAssignment**(β', FR', m, U)
10: **if** $feas$ **then**
11: **return** $(\alpha \cup \{(R_k, L_l)\}, true)$
12: **end if**
13: **end if**
14: **end if**
15: **end for**
16: **if** $(\beta, FR') \notin U$ **then**
17: $(\alpha, feas) \leftarrow$ **FindResAssignment**(β, FR', m, U)
18: **if** $feas$ **then**
19: **return** $(\alpha, true)$
20: **end if**
21: **end if**
22: **end if**
23: $U \leftarrow U \cup \{(\beta, FR)\}$
24: **return** $(\{\}, false)$

(β', FR'), where $(\beta', FR') \in U$ means that it is not possible to satisfy the demand vector β' with the free resources FR'. If it is possible to find a resource assignment using resources in FR that covers the skill requirements β, the algorithm returns $(\alpha, true)$. Here α is a resource assignment expressed as a set of pairs (R_k, L_l), meaning that resource R_k will perform skill L_l. If it is not possible to find an assignment satisfying the skill demands, the algorithm returns $(\{\}, false)$. The auxiliary method $getFirst(FR)$ returns the resource of FR that first occurs in the list of all resources $R_1, \ldots, R_v$, hence the backtrack tree explores the resources in the same order in all paths. The auxiliary method $decreaseBy1(\beta, l)$ returns β but replacing b_l by $b_l - 1$. Algorithm 9 follows a backtracking scheme that explores all the possibilities. Each call of the algorithm considers a resource, that is either assigned to a skill (line 9), or not used (line 17). By means of dynamic programming the algorithms avoids to look twice for an assignment of a particular resource set FR and skill demand vector β. This is achieved by storing into set U the already tried pairs (β, FR). This way, the algorithm is called only once for each possible pair, and the number of calls is:

$$O(|FR| \cdot \prod_{l=1}^{s}(b_l + 1))$$

This bound is acceptable for the instances that are currently tackled with exact solving methods, and therefore we use Algorithm 9 as an oracle in the PSGS method. More precisely, when we try to place the start of an activity A_i at time t, we call Algorithm 9 with $\beta = b_{i,1}, \ldots, b_{i,s}$, with FR being the subset of the resources that are not yet occupied at time t, and U being the empty set in the first call. The contents of U are kept between different calls.

6.4.4 Results

In this section we evaluate our formulation using the same sets that were used in [YFS17] and that the authors made publicly available. There are 5 datasets with the following name and properties:

set-1a: It contains 216 instances with 20 non-dummy activities, 4 skills and between 10 and 30 resources.

set-1b: It contains 216 instances with 40 non-dummy activities, 4 skills and between 20 and 60 resources.

set-2a: It contains 110 instances with between 18 and 49 non-dummy activities, between 2 and 8 skills, and between 5 and 14 resources.

set	tool	Q1	med.	Q3	avg.	t.o.
set-1a	LCG	0.43	0.59	1.46	3.48	0
	SMT	0.16	0.27	0.50	4.08	0
set-1b	LCG	t.o.	t.o.	t.o.	564.63	199
	SMT	51.41	t.o.	t.o.	384.23	123
set-2a	LCG	2.08	22.30	t.o.	244.31	40
	SMT	0.37	4.76	t.o.	195.87	30
set-2b	LCG	0.55	3.44	287.85	161.33	18
	SMT	0.13	1.29	16.44	90.00	10
set-2c	LCG	0.24	0.62	2.90	7.02	0
	SMT	0.09	0.24	0.64	10.44	1

Table 6.5: Solving time results for the instances of MSPSP. All values are in seconds. We give a timeout of 600 seconds to each execution. The first column contains the name of the dataset. Column 2 specifies, for each row, the tool used to solve the instances of the dataset. Columns *Q1*, *med.* and *Q3* specify the first, second, and third quartile of the running times of a dataset (*t.o.* means timeout for that quartile). Column *avg.* contains the average run time, counting timeouts as 600 seconds. The last column contains the number of instances that timed out without having found an optimal solution and proven its optimality.

set-2b: It contains 77 instances with between 30 and 60 non-dummy activities, between 9 and 15 skills, and between 5 and 19 resources.

set-2c: It contains 91 instances with between 20 and 30 non-dummy activities, between 3 and 12 skills, and between 4 and 15 resources.

Sets *set-1a* and *set-1b* were originally created in [CLSdG12, ACSdG16], but although we did our best to contact the authors and obtain the datasets, we did not succeed. We have also been unable to obtain their exact solvers. The authors of [YFS17] had the same difficulties, and they reproduced the datasets from scratch using the same parameters, and these are the sets that we use in our experiments. Sets *set-2a*, *set-2b* and *set-2c* were originally created in [MBMPR14].

We compare our system with the one presented in [YFS17], which has shown to be the best exact system for these sets. It consists of a CP model which is solved with the LCG solver *chuffed*. That system reports the best results when using a variable decision strategy named *priority_search*, which allows them to organise the variables by groups. For each activity, they create a group containing a variable representing its start time, and also all the

variables representing the resource assignments for that activity. Then, the search procedure assigns all the variables in a same group consecutively before branching on variables of another group. We will refer to this system as *LCG*, and to our system as *SMT*.

We have run both *LCG*, and *SMT* on identical machines. It is a 8GB Intel® Xeon® E3-1220v2 machine at 3.10 GHz. In all experiments we use Yices 2.6.1 [DdM06a] as the core SMT solver. The results are presented in Table 6.5. It can be appreciated our system is able to solve a substantial number of instances more than *LCG*, and the percentiles and average of the solving time are also smaller in most cases. The improvement in *set-1b* is especially significant, and this set turns to be the hardest one. *SMT* is able to solve a considerable number of instances more than *LCG*. Only in *set-2c* *LCG* has solved one instance more than *SMT*, although *SMT* has a smaller third quartile solving time.

6.5 Chapter Summary

In this chapter we have proposed methods and SMT formulations to efficiently solve different RCPSP extensions. Each of the tackled problems presents additional constraints which we can naturally handle with our formulations. For the MRCPSP we deal with multiple execution modes by representing each mode choice with a Boolean variable. Then, this variable is used to define conditional values on the precedence constraints and identify what is the effective resource consumption of each activity. For the RCPSP/t, the Time-indexed approach let us naturally deal with the time-dependency of resource availabilities and requests. Regarding MRCPSP/max, the generalised precedences can be defined by setting the appropriate values (possibly negative) in IDL expressions that specify the minimum time difference between the starts of two activities. For the MSPSP, we formulate the resource constraints by introducing many variables with combined semantics. Using these variables, we are able to identify what resources are used by every activity, and whether a resource is performing a particular activity at a particular time instant.

One of the core components of all formulations is the use of SAT encodings of PB(AMO) constraints. We use the precedence relations to detect incompatibilities between activities, and from there we can detect AMO constraints. In the different considered problems, there appear further AMO constraints, due to having multiple execution modes or due to the variability of requests over time.

We have provided solutions to perform a quick computation of an upper bound for the makespan, in most cases based on the PSGS method. Comput-

ing a good upper bound let us mitigate the potential drawback of Time-indexed formulations, that is having long scheduling horizons. It is particularly relevant the computation of the upper bound of the MSPSP, for which we provide an algorithm to find resource assignments to activities that can be integrated in PSGS. The only case in which we do not use PSGS is the MRCPSP/max, where the maximal time lags make the method not suitable. Instead, in MRCPSP/max we first compute a lower bound of the makespan, by solving a relaxation of the problem which is not penalised by long scheduling horizons. Then we compute an upper bound by solving an over-constrained problem. This method let us optimally solve a large number of the existing benchmark instances without the need of solving an SMT formula representing the whole original problem. Moreover we provide new and more challenging datasets for the MRCPSP/max, which are not optimally solvable only with relaxations.

We evaluate our systems experimentally by solving the most relevant benchmark instances of the different problems from the literature. Also, we compare with state-of-the-art exact solvers for each one of the problems. Our results show that in most cases our systems give the best performance, both in solving times and number of solved instances.

Chapter 7

New SAT Encodings of PB(AMO) Constraints

There exist many works on encoding PB constraints to SAT that are not based in decision diagrams [PS15]. The most powerful are based on Sequential Weight Counters [HMS12], Generalized Totalizers [JMM15], and Polynomial Watchdog schemes [BBR09, MPS14]. In these works it is shown that in some occasions these encodings can perform better than BDD-based ones. In Chapter 4 we proposed an MDD-based SAT encoding of PB constraints under the assumption that there exist some AMO relations on disjoint subsets of variables, i.e. a specialised encoding for the PB constraint in a PB(AMO) constraints. These results encourage us to address the question of whether other SAT encodings of PB constraints not based on decision diagrams can be improved in the presence of AMOs. In this chapter we revisit many state-of-the-art SAT encodings of PB constraints and propose improved versions of those encodings for PB(AMO) constraints. More precisely, we provide modifications of the Sequential Weight Counter, Generalized Totalizer, and Global Polynomial Watchdog encodings for encoding PB(AMO) constraints. We also show that the new encodings preserve the propagation properties that the original ones have for encoding PBs. Our experimental results show again that the size of the SAT encodings of PB constraints can be dramatically reduced thanks to taking the AMO constraints into account, and that there can be a huge time performance improvement when using the new encodings. We provide datasets which contain AMO constraints and PB constraints with different configurations, and we show that some encodings are better than others for particular kinds of PB.

The work presented in this chapter has been published in [BCSV19]. This

work is related to the objective number 4 of this thesis. The rest of this chapter is organised as follows:

- Section 7.1 recalls the general idea of our encoding approach, and specifies some assumptions that we make.
- In Section 7.2 we introduce the Generalized Sequential Weight Counter encoding.
- In Section 7.3 we introduce the Generalized Generalized Totalizer encoding.
- In Section 7.4 we introduce the Generalized Global Polynomial Watchdog encoding, and explain how to adapt it to the Generalized Binary Merger Encoding.
- Section 7.5 contains an experimental evaluation of the proposed encodings.
- In Section 7.6 we summarise the contributions of this chapter.

7.1 Encoding Idea and Assumptions

Similarly to what we did in Chapter 4, the encodings that we propose in this chapter encode PB(AMO)s of the form $P \wedge M_1 \wedge \cdots \wedge M_N$ in a combined way. On the one hand we encode the conjunction of AMO constraints in the usual way, i.e. we encode each M_i separately and use the conjunction of all the resulting clauses. On the other hand we encode the PB constraint P assuming that the accompanying AMO constraints are already enforced in some way. This is precisely what will let us reduce the size of the encoding of the PB constraint. We do not restrict to a particular encoding for the AMO constraints. Even more, in the context of a bigger formula, if the AMO constraints are logically implied by the formula at hand, then the encoding of the PB constraint will suffice to obtain a correct encoding of the PB(AMO) constraint (see Lemma 4.1.3).

We start each of the following sections giving an intuitive explanation of an already existing encoding of PB constraints. Then, we propose a generalized version of it in order to encode PB(AMO) constraints. Since PB(AMO) constraints generalize PB constraints, we follow the convention of naming the new encodings after the original encoding, prefixing them with the word *Generalized*, e.g., from the Sequential Weight Counter (SWC) encoding we provide the Generalized Sequential Weight Counter (GSWC) encoding.

We make the following assumptions on the PB(AMO) constraint $P \wedge M_1 \wedge \cdots \wedge M_N$ at hand: (i) unless otherwise stated, P is of the form $\sum_{i=1}^{n} q_i x_i \leq K$, with $q_i \geq 0$, i.e. it is monotonic decreasing (w.l.o.g., see Section 4.4.2); (ii) P is neither trivially true nor false (i.e., we assume that $0 \leq K < \sum_{i=1}^{n} q_i$); (iii) no variable is trivially removable (i.e., $0 < q_i \leq K$).

Our goal is to encode a PB(AMO) constraint of the form $P \wedge M_1 \wedge \cdots \wedge M_N$ in the already mentioned way, that is generating a set of clauses that behave as an encoding of P for assignments satisfying the AMOs $M_1, \ldots, M_N$. Therefore, similarly to what we do with the Minimal Encoding in Chapter 4, for each one of the new encodings we receive as input the PB constraint P and a partition $\mathcal{X} = \{X_1, \ldots, X_N\}$ of the variables of P, such that $X_i = \mathit{scope}(M_i)$.

7.2 Sequential Weight Counter Encoding

The *Sequential Weight Counter* (SWC) encoding for PB constraints was introduced in [HMS12]. The idea is to encode the PB constraint by a circuit that sequentially sums from left to right the coefficients (a.k.a. weights) q_i whose variable x_i is set to true. Specifically, given a PB constraint $\sum_{i=1}^{n} q_i x_i \leq K$, there is a sequence of n counters of K inputs and K outputs, where the i-th counter is associated to variable x_i. Each counter receives as input a vector of Boolean variables, which is the unary representation of an integer value, and adds the weight q_i to the output, which is also a unary representation, if the associated variable x_i is set to true. Therefore, the i-th counter receives as input $\sum_{j=1}^{i-1} q_j x_j$ and outputs $\sum_{j=1}^{i} q_j x_j$. Note that the output of the counter number $i-1$ is the input of the i-th counter.

An example of a sequence of counters is shown in Figure 7.1. The encoding introduces $n \cdot K$ variables, denoted $s_{i,j}$, with $1 \leq i \leq n$, $1 \leq j \leq K$, where $s_{i,j}$ is the j-th output of the i-th counter and also the j-th input of the $(i+1)$-th counter. The encoding introduces the clauses:

$$\overline{s_{i-1,j}} \vee s_{i,j} \qquad 2 \leq i < n, 1 \leq j \leq K \tag{7.1}$$

$$\overline{x_i} \vee s_{i,j} \qquad 1 \leq i < n, 1 \leq j \leq q_i \tag{7.2}$$

$$\overline{s_{i-1,j}} \vee \overline{x_i} \vee s_{i,j+q_i} \qquad 2 \leq i < n, 1 \leq j \leq K - q_i \tag{7.3}$$

$$\overline{s_{i-1,K+1-q_i}} \vee \overline{x_i} \qquad 2 \leq i \leq n \tag{7.4}$$

where $s_{0,j}$ is the constant 0 for all j, to represent the input of the first counter which is the empty sum. Clauses (7.1) state that $\sum_{j=1}^{i} q_j x_j \geq \sum_{j=1}^{i-1} q_j x_j$. Clauses (7.2) and (7.3) enforce that if a variable x_i is true then its coefficient is added to the input of the next counter. Finally, Clauses (7.4) enforce that the sum never exceeds K. Note that the output of the last circuit is not defined

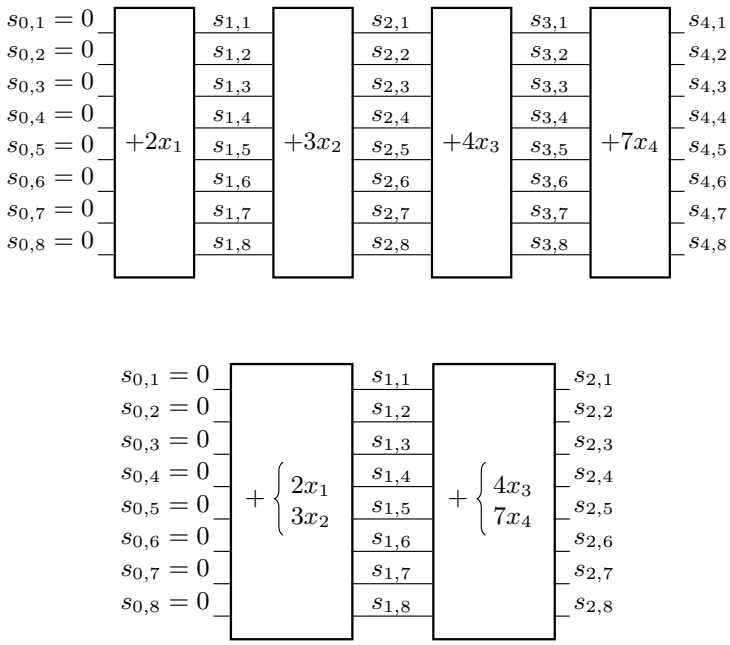


Figure 7.1: At the top: high level circuit representation of $SWC(2x_1 + 3x_2 + 4x_3 + 7x_4 \leq 8)$. At the bottom: high level circuit representation of $GSWC(2x_1 + 3x_2 + 4x_3 + 7x_4 \leq 8, \{\{x_1, x_2\}, \{x_3, x_4\}\})$.

by Clauses (7.1), (7.2) and (7.3) —i.e. clauses are not added for $i = n$— since this is not necessary to encode the PB constraint. However, such clauses can be added if it is desired to get a unary representation of $\sum_{i=1}^{n} q_i x_i$ as the output of the circuit.

7.2.1 Generalized Sequential Weight Counter (GSWC)

We define the GSWC encoding by, instead of associating a single product $q_i x_i$ from the PB constraint to each counter, associating a set of products to each of them. In our generalization, given a partition $\mathcal{X} = \{X_1, \ldots, X_N\}$ of the variables of the PB constraint, the resulting formulation will have just N counters, where the i-th counter will handle all the products $q_l x_l$ for the variables x_l in X_i. If the variables in each set X_i are subject to an AMO constraint then, given an assignment satisfying those constraints, at most one coefficient q_l will be added by each counter, and the output of the whole circuit will correspond to the value of the left hand side sum of the PB constraint, i.e., $\sum_{i=1}^{n} q_i x_i$. Analogously as in the original encoding, we will enforce that it is not reached a sum that exceeds K. The GSWC encoding introduces the

following clauses:

$$\overline{s_{i-1,j}} \vee s_{i,j} \qquad 2 \leq i < N,\ 1 \leq j \leq K \tag{7.5}$$

$$\overline{x_l} \vee s_{i,j} \qquad 1 \leq i < N,\ x_l \in X_i,\ 1 \leq j \leq q_l \tag{7.6}$$

$$\overline{s_{i-1,j}} \vee \overline{x_l} \vee s_{i,j+q_l} \qquad 2 \leq i < N,\ x_l \in X_i,\ 1 \leq j \leq K - q_l \tag{7.7}$$

$$\overline{s_{i-1,K+1-q_l}} \vee \overline{x_l} \qquad 2 \leq i \leq N,\ x_l \in X_i \tag{7.8}$$

Clauses (7.5) propagate the accumulated sum in the same way as Clauses (7.1). Clauses (7.6) and (7.7) enforce $S_i \geq S_{i-1} + q_l x_l$, for all $x_l \in X_i$, where S_{i-1} and S_i are respectively the input and output value of the i-th counter. Clauses (7.8) enforce that the sum never exceeds K. A high level circuit representation of a GSWC encoding is shown in Figure 7.1.

The main difference between the SWC and GSWC encodings is that the latter has only N counters, instead of n, and therefore introduces less fresh variables (assuming $N < n$). Also, when $N < n$ the number of Clauses (7.5) in the GSWC encoding is smaller than the number of Clauses (7.1) in the SWC encoding. The SWC encoding requires $O(nK)$ auxiliary variables and $O(nK)$ clauses, while the GSWC encoding requires $O(NK)$ auxiliary variables and $O(nK)$ clauses.

By $GSWC(P, \mathcal{X})$ we denote the set of clauses derived from a PB constraint P and a partition $\mathcal{X}$, as described above.

Lemma 7.2.1. *Let $\mathcal{P}$ be a PB(AMO) of the form $P \wedge M_1 \wedge \cdots \wedge M_N$, with P of the form $\sum_{i=1}^{n} q_i x_i \leq K$, and $\mathcal{X} = \{X_1, \ldots, X_N\}$ be a partition of the variables of P, such that $X_i = scope(M_i)$. The conjunction of $GSWC(P, \mathcal{X})$ with an encoding of $M_1 \wedge \cdots \wedge M_N$ is an encoding of $\mathcal{P}$.*

In [HMS12] it is proved that the SWC encoding UP-maintains GAC. The GSWC encoding preserves this property.

Theorem 7.2.2. *Let $\mathcal{P}$ be a PB(AMO) of the form $P \wedge M_1 \wedge \cdots \wedge M_N$, with P of the form $\sum_{i=1}^{n} q_i x_i \leq K$, and $\mathcal{X} = \{X_1, \ldots, X_N\}$ be a partition of the variables of P, such that $X_i = scope(M_i)$. The conjunction of $GSWC(P, \mathcal{X})$ with an UP-maintaining GAC encoding of $M_1 \wedge \cdots \wedge M_N$ is UP-maintaining GAC.*

Proof. Let S denote the conjunction of $GSWC(P, \mathcal{X})$ with a UP-maintaining GAC encoding of $M_1 \wedge \cdots \wedge M_N$. Let A be a partial assignment of the variables of S, which is extendible to a satisfying assignment of $\mathcal{P}$. Therefore, no AMO constraint M_i is violated under A. We need to show that for every variable x of $\mathcal{P}$ such that x is not assigned in A, if $A \cup \{x\}$ cannot be extended to a satisfying

assignment of $\mathcal{P}$, then x is set to false by unit propagating A on S (note that $A \cup \{\overline{x}\}$ can always be extended to a satisfying assignment, so we don't need to consider this case). W.l.o.g., assume that $x_1 \in X_1$ is such variable. If $A \cup \{x_1\}$ cannot be extended to a satisfying assignment of $M_1 \wedge \cdots \wedge M_N$ then, by the assumption that S contains an UP-maintaining GAC encoding of $M_1 \wedge \cdots \wedge M_N$, we have that x_1 is set to false by unit propagation. Assume now the contrary, i.e., that $A \cup \{x_1\}$ can be extended to an assignment satisfying the AMOs. In this case, the reason why UP should set x_1 to false is that $A \cup \{x_1\}$ cannot be extended to satisfy P. Since $A \cup \{x_1\}$ does not violate $M_1 \wedge \cdots \wedge M_N$, at most one variable in X_i is true in A, for $2 \leq i \leq N$, and no variable in X_1 is true in A. Let us construct a PB constraint P' from P by picking one variable x_{j_i} from each set X_i, $2 \leq i \leq N$, as follows: if X_i contains a variable which is true in A, then this is the variable to be picked up from X_i, otherwise pick up any variable. We define $P' : q_1 x_1 + \sum_{i=2}^{N} q_{j_i} x_{j_i} \leq K$. Since P' contains all the variables which are true in A, and due to the monotonicity of P, we have that $q_1 x_1 + \sum_{i=2}^{N} q_{j_i} x_{j_i}$ is equisatisfiable to $\sum_{i=1}^{n} q_i x_i$ under the assignment $A \cup \{x_1\}$. Therefore $A \cup \{x_1\}$ can neither be extended to a model of P'. It is not hard to see that $GSWC(P, \mathcal{X})$ contains all the clauses of $SWC(P')$, and it is already proved that the SWC encoding UP-maintains GAC. Therefore, all the clauses required to set x_1 to false by UP are contained in S. □

7.3 Generalized Totalizer Encoding

The *Generalized Totalizer* (GT) encoding was presented in [JMM15] as a generalization of the Totalizer encoding for cardinality constraints [BB03]. The overall idea of GT is to represent a PB constraint $\sum_{i=1}^{n} q_i x_i \leq K$ as a binary tree. Every node of the tree has associated a distinct label and an attribute *vars* which consists of a set of Boolean variables. Each variable x_i of the PB constraint is placed into the attribute *vars* of a different leaf node, and is renamed after the label of the node and its associated coefficient q_i (e.g., given the product $3x_1$, if the variable x_1 is inserted into a leaf node labelled by letter O, then the variable is named o_3). The attribute *vars* of any non-leaf node labelled O contains a variable o_w iff there is a subset of the underlying leaves which sums exactly w, for values of w in the range $[1, K]$, taking i for the value of each leaf node L with variable l_i. Also, *vars* contains a variable o_{K+1} iff any of the sums is larger than K. Figure 7.2 illustrates the binary tree of a PB constraint.

The clauses of the encoding enforce that each non-leaf variable o_w is set to true if the underlying variables which sum w (or more than K for $w = K + 1$)

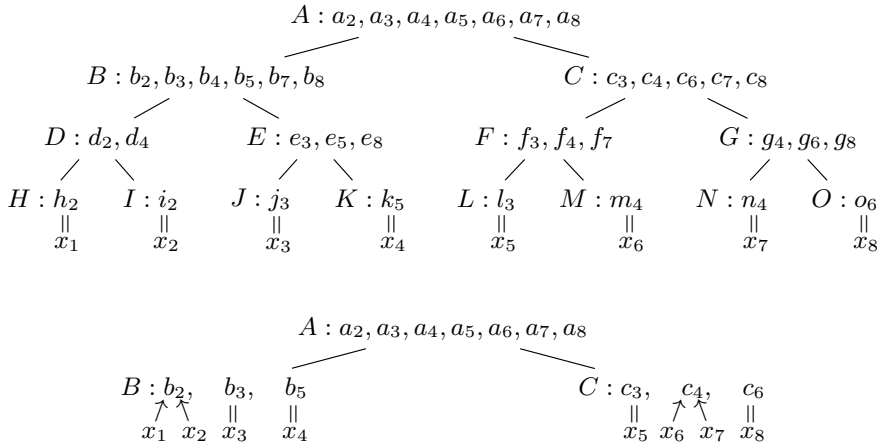


Figure 7.2: At the top: binary tree of $GT(2x_1 + 2x_2 + 3x_3 + 5x_4 + 3x_5 + 4x_6 + 4x_7 + 6x_8 \leq 7)$. At the bottom: binary tree of $GGT(2x_1 + 2x_2 + 3x_3 + 5x_4 + 3x_5 + 4x_6 + 4x_7 + 6x_8 \leq 7, \{\{x_1, x_2, x_3, x_4\}, \{x_5, x_6, x_7, x_8\}\})$.

are set to true. Moreover it is enforced, at the root node, that the variable representing a sum larger than K is false. The GT encoding introduces the following clauses for each non-leaf node O with children L and R:

$$\overline{l_{w_1}} \vee \overline{r_{w_2}} \vee o_{w_3} \qquad l_{w_1} \in L.vars,\ r_{w_2} \in R.vars,\ w_3 = \min(w_1 + w_2, K + 1) \tag{7.9}$$

$$\overline{t_w} \vee o_w \qquad t_w \in L.vars \cup R.vars \tag{7.10}$$

It also introduces the unary clause

$$\overline{a_{k+1}} \tag{7.11}$$

where A is the root node of the tree and $a_{k+1} \in A.vars$. Note that variable a_{k+1} will exist, since otherwise the constraint would be trivially satisfied. Clauses (7.9) enforce that the variable o_{w_3} will be set to true by UP if there exists a pair of variables l_{w_1}, r_{w_2} from the children nodes that are set to true and such that $w_3 = \min(w_1 + w_2, K + 1)$. Clauses (7.10) enforce that the variable o_w will be set to true by UP if some child has a variable t_w set to true. Finally, Clause (7.11) states that the sum of the tree (i.e., the value of the left hand side expression of the PB constraint) cannot be greater than K.

7.3.1 Generalized Generalized Totalizer (GGT)

In our generalization of the GT encoding, we will use the same definition of the binary tree, but the leafs will be instantiated differently. Instead of introducing

a leaf node for each variable of the PB constraint, what we do is to introduce a leaf node for each of the sets in the partition $\mathcal{X}$. The leaf node O associated to set X_i will contain a variable o_{q_l} in its *vars* attribute for each different coefficient q_l such that $x_l \in X_i$. If there is a single occurrence of coefficient q_l, then x_l is renamed as o_{q_l} and placed in $O.vars$, as in the GT encoding. If there are multiple occurrences of a coefficient q_l, we introduce a fresh variable o_{q_l}. The following clauses relate the fresh leaf variables with the variables of the PB constraint:

$$\overline{x_l} \vee o_{q_l} \qquad X_i \in \mathcal{X},\, x_l \in X_i,\, |\{x_{l'} \in X_i \mid q_{l'} = q_l\}| \geq 2 \tag{7.12}$$

Now the attribute *vars* of the leaf nodes can contain more than one variable, and the attribute *vars* of a non-leaf node O contains a variable o_w iff some combination of the leaf nodes, taking at most one variable of each leaf node, sums exactly w —again, only for values of w in the range $[1, K]$, taking i for the value of a varialbe l_i in a leaf node L.

The GGT encoding introduces Clauses (7.9), (7.10) and (7.11) as in the GT encoding, and Clauses (7.12). Figure 7.2 depicts the binary tree of a GGT encoding. Note that assuming that an AMO constraint over each set X_i is satisfied, at most one of the variables in each leaf node will be true, and therefore the encoding correctly evaluates $\sum_{i=1}^{n} q_i x_i \leq K$. GT encoding requires $O(nK)$ auxiliary variables and $O(nK^2)$ clauses, while GGT encoding requires $O(NK)$ auxiliary variables and $O(NK^2)$ clauses.

By $GGT(P, \mathcal{X})$ we denote the set of clauses derived from a PB constraint P and a partition $\mathcal{X}$, as described above.

Lemma 7.3.1. *Let $\mathcal{P}$ be a PB(AMO) of the form $P \wedge M_1 \wedge \cdots \wedge M_N$, with P of the form $\sum_{i=1}^{n} q_i x_i \leq K$, and $\mathcal{X} = \{X_1, \ldots, X_N\}$ be a partition of the variables of P, such that $X_i = scope(M_i)$. The conjunction of $GGT(P, \mathcal{X})$ with an encoding of $M_1 \wedge \cdots \wedge M_N$ is an encoding of $\mathcal{P}$.*

In [JMM15] it is proved that the GT encoding UP-maintains GAC. The GGT encoding preserves this property.

Theorem 7.3.2. *Let $\mathcal{P}$ be a PB(AMO) of the form $P \wedge M_1 \wedge \cdots \wedge M_N$, with P of the form $\sum_{i=1}^{n} q_i x_i \leq K$, and $\mathcal{X} = \{X_1, \ldots, X_N\}$ be a partition of the variables of P, such that $X_i = scope(M_i)$. The conjunction of $GGT(P, \mathcal{X})$ with an UP-maintaining GAC encoding of $M_1 \wedge \cdots \wedge M_N$ is UP-maintaining GAC.*

The proof is analogous to the proof of Theorem 7.2.2.

7.4 Global Polynomial Watchdog Encoding

The *Global Polynomial Watchdog* (GPW) encoding was presented in [BBR09]. It uses as basis a *polynomial watchdog* formula, denoted by $PW(P)$, which is associated with a PB constraint P. The formula $PW(P)$ has a variable named the *output* variable, denoted w, which is set to 1 by UP as soon as P is falsified.

The GPW encoding is defined for PB constraints of the form $\sum_{i=1}^{n} q_i x_i < K$, i.e., with a strict inequality instead of a non-strict one. The first step is to normalise the constraint to the form $T + \sum_{i=1}^{n} q_i x_i < m2^p$, where p, T and m are defined as follows: $p = \lfloor \log_2(\max_{i=1..n}(q_i)) \rfloor$ is the index of the most significant bit in the binary representation of the largest coefficient q_i, being 0 the index of the least significant bit. In other words, $p+1$ is the number of bits needed to represent q_i in binary notation; T is the smallest non-negative integer such that $K + T$ is a multiple of 2^p; $m = (K + T)/2^p$.

Once the constraint is expressed in this form, it is computed a set B_r of variables of P (called *bucket*) for each bit $0 \leq r \leq p$. We denote by $b_r(q_i)$ the r-th bit of the binary representation of the integer q_i. Bucket B_r contains all the variables x_i such that $b_r(q_i) = 1$. Bucket B_r also contains a 1 constant if $b_r(T) = 1$.

Example 5. *The following is the transformation to apply to the PB constraint* $2x_1 + 3x_2 + 4x_3 + 7x_4 < 9$. *We have that* $p = 2$, *and* $T = 3$ *is the smallest integer such that* $T + K = 12$ *is a multiple of* 2^p, *with* $m = 3$. *Therefore, the constraint is expressed as* $3 + 2x_1 + 3x_2 + 4x_3 + 7x_4 < 12$. *The content of buckets* B_0, B_1 *and* B_2 *is illustrated in Figure 7.3.*

The idea is to decompose each coefficient in its binary representation and sum each bit having the same weight.

The formula $PW(P)$ can be represented as a circuit, as can be seen in Figure 7.3 corresponding to Example 5. We denote by $\langle B_r \rangle$ a vector with an arbitrary order containing the elements of bucket B_r. The formula $PW(P)$ uses two main components: the formulas $\phi(V)$ and $\psi(V_1, V_2)$. The formula $\phi(V)$ has as input a vector of Boolean variables V, and has as output a vector of $|V|$ variables named $U(V)$. The formula $\phi(V)$ enforces that $U(V)$ is the unary representation of the sum of the input variables. The formula $\psi(V_1, V_2)$, has as input two vectors of variables V_1 and V_2, which are the unary representation of two integers, and has as output a vector of $|V_1| + |V_2|$ variables named S. The formula $\psi(V_1, V_2)$ enforces that S is the unary representation of $V_1 + V_2$. In the definition of $PW(P)$, we denote by S_r the output of the ψ formula related with bucket B_r, for $1 \leq r \leq p$, and we define $S_0 = U(\langle B_0 \rangle)$. Half of the value

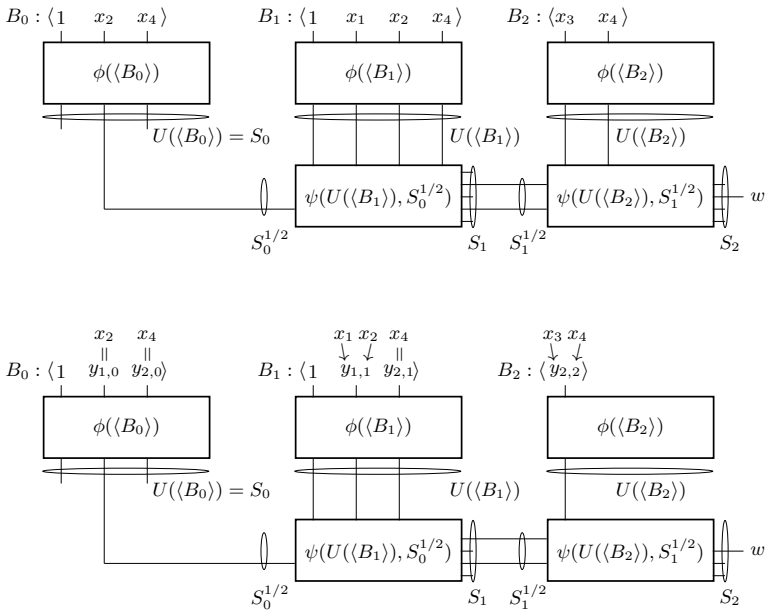


Figure 7.3: At the top: circuit representation of $PW(2x_1 + 3x_2 + 4x_3 + 7x_4 < 9)$. At the bottom: circuit representation of $PW(2x_1 + 3x_2 + 4x_3 + 7x_4 < 9, \{\{x_1, x_2\}, \{x_3, x_4\}\})$.

of S_k, for a weight 2^k, is computed with operator $\frac{1}{2}$ and integrated in the sum for weight 2^{k+1}. Then, the formula $PW(P)$ is defined as the conjunction of these two formulas:

$$\phi(\langle B_r \rangle) \qquad 0 \leq r \leq p \tag{7.13}$$

$$\psi(U(\langle B_r \rangle), S_{r-1}^{1/2}) \qquad 1 \leq r \leq p \tag{7.14}$$

The GPW encoding is then defined as

$$PW(P) \wedge \overline{w} \tag{7.15}$$

where $PW(P)$ encodes ϕ with a totalizer, and ψ with an adder of unary numbers. For further correctness proofs and detailed explanations we refer the reader to the original work introducing Polynomial Watchdogs [BBR09]. The basic idea is that the m-th bit of S_p, represented with variable w, is set to 1 by UP if the sum of the constraint is greater or equal than $m2^p = T + K$. If w is set to 1 the encoding is not satisfied.

7.4.1 Generalized Global Polynomial Watchdog (GGPW)

We define the GGPW encoding by using a *generalized polynomial watchdog formula* $PW(P, \mathcal{X})$ instead of the original polynomial watchdog formula. Again, P has to be normalised to the form $T + \sum_{i=1}^{n} q_i x_i < m2^p$ in the same way as in $PW(P)$. For each set X_i, $PW(P, \mathcal{X})$ will contain a vector of variables $Y_i = \langle y_{i,p}, y_{i,p-1}, \ldots, y_{i,0} \rangle$.

Y_i is interpreted as binary number, where (at least) the bits corresponding to the binary representation of q_l, for all $x_l \in X_i$ such that x_l is true, are set to one. Therefore, when exactly one x_l is true, Y_i will be greater than or equal to q_l. The following clauses define the variables Y_i:

$$\overline{x_l} \vee y_{i,r} \qquad 0 \leq r \leq p,\ 1 \leq i \leq N,\ x_l \in X_i,\ b_r(q_l) = 1 \tag{7.16}$$

In this case the bucket B_r, for each bit $0 \leq r \leq p$, will contain the variables $y_{1,r}, y_{2,r}, \ldots, y_{N,r}$. Bucket B_r will also contain a 1 constant if $b_r(T) = 1$.

The formula $PW(P, \mathcal{X})$ is defined as the conjunction of (7.13), (7.14) and (7.16). Some considerations can be taken into account on Clauses (7.16) in order to optimise the encoding:

- If there is no $x_l \in X_i$ such that $b_r(q_l) = 1$, and therefore the variable $y_{i,r}$ does not appear in any clause of (7.16), then this variable is not created nor included in any bucket.
- If there is only one variable $x_l \in X_i$ such that $b_r(q_l) = 1$, then the variable $y_{i,r}$ is the variable x_l itself, and Clause (7.16) is not added for $y_{i,r}$.
- Otherwise, $y_{i,r}$ is indeed a fresh variable and Clause (7.16) is added.

Figure 7.3 contains a circuit representation of $PW(P, \mathcal{X})$. The GGPW encoding is defined as

$$PW(P, \mathcal{X}) \wedge \overline{w} \tag{7.17}$$

where $PW(P, \mathcal{X})$ encodes ϕ with a totalizer, and ψ with an adder of unary numbers. Similarly as in the other newly introduced encodings, given an assignment that satisfies an AMO constraint over each $X_i \in \mathcal{X}$, this encoding represents the PB constraint $\sum_{i=1}^{n} q_i x_i < K$ in a more compact way.

The GPW encoding introduces $O(n \log(n) \log(q_{max}))$ fresh variables and $O(n^2 \log(n) \log(q_{max}))$ clauses, while the GGPW encoding introduces $O(N \log(N) \log(q_{max}))$ fresh variables and $O(N^2 \log(N) \log(q_{max}))$ clauses,

where $q_{max} = \max_{i=1}^{n} q_i$. This follows from the fact that a totalizer ϕ with n input variables requires $O(n \log(n))$ auxiliary variables and $O(n^2 \log(n))$ clauses, and an adder ψ of unary numbers with n input variables requires $O(n)$ auxiliary variables and $O(n^2)$ clauses; see [BBR09].

Lemma 7.4.1. *Let $\mathcal{P}$ be a PB(AMO) of the form $P \wedge M_1 \wedge \cdots \wedge M_N$, with P of the form $\sum_{i=1}^{n} q_i x_i < K$, and $\mathcal{X} = \{X_1, \ldots, X_N\}$ be a partition of the variables of P, such that $X_i = scope(M_i)$. The conjunction of $GGPW(P, \mathcal{X})$ with an encoding of $M_1 \wedge \cdots \wedge M_N$ is an encoding of $\mathcal{P}$.*

In [BBR09] it is shown that the GPW encoding does not UP-maintain GAC. As stated earlier, a PB(AMO) constraint with only AMO constraints of one variable is indeed a PB constraint. In this case the GGPW and GPW encodings would be identical. Therefore, the GGPW encoding does neither UP-maintain GAC.

7.4.2 Generalized Binary Merger (GBM)

The *Binary Merger* (BM) encoding was introduced in [MPS14]. This encoding is essentially another implementation of the GPW encoding. The difference in BM encoding is that the formulas ϕ and ψ are respectively implemented using sorters and odd-even mergers [ANOR11]. This way, the BM encoding is asymptotically smaller in the number of clauses and slightly bigger in the number of variables than the GPW encoding. The BM encoding can be generalized to the Generalized Binary Merger encoding, in order to deal with PB(AMO) constraints, in the same way as GPW encoding. However, we have not observed significant differences between the BM and GPW based encodings, and therefore we do not provide detailed results for BM encoding.

7.5 Results

In this section we report a clean comparison between the different encodings for PB(AMO) constraints, and also between those and the classical encodings for PB constraints. For this purpose, we craft and use benchmark sets of problems consisting on conjunctions of AMO constraints and PB constraints. Each instance is defined by four parameters which have a direct influence on the size complexity of the encodings: L is the number of PB constraints, N is the number of AMO constraints, M is the number of Boolean variables in each AMO constraint, and Q is the maximum coefficient of a variable in a PB constraint. The variables of the AMO constraints will be disjoint, so there is a total of $n = N \cdot M$ Boolean variables in each instance. The PB

constraints contain all n variables. The j-th variable in the i-it AMO constraint is named $x_{i,j}$. The coefficients in the PB constraints are generated uniformly and independently at random in the range $[1, Q]$. The resulting instance has the following constraints:

$$\sum_{i=1}^{N}\sum_{j=1}^{M} q_{i,j,k} \cdot x_{i,j} \leq K_k \qquad 1 \leq k \leq L \tag{7.18}$$

$$\sum_{j=1}^{M} x_{i,j} \leq 1 \qquad 1 \leq i \leq N \tag{7.19}$$

$$\sum_{j=1}^{M} x_{i,j} \geq 1 \qquad 1 \leq i \leq N \tag{7.20}$$

The conjunction of PB and AMO constraints (7.18) and (7.19) is not a hard problem, since a trivial solution is to set all the variables $x_{i,j}$ to 0. For this reason we add *at-least-one* Constraints (7.20), which require that at least one variable in each AMO group is set to true. Essentially, this set of constraints is a decision version of the Multi-Choice Multidimensional Knapsack Problem (MMKP), which is NP-complete. Therefore, we have generated the benchmarks using the MMKP instance generator from [HLS10].

We provide three different datasets with different parameters, with the aim of showing which encoding is better suited for each kind of PB constraint. The instances in a dataset are distributed in families, and every family has values of K_k randomly distributed around a different mean in the range $[1, M \cdot Q]$. The values of K_k are proportional to the values of the coefficients, because otherwise the PB constraints would be trivially satisfied or unsatisfied. We choose different values of K_k to ensure that in the datasets there are instances of different hardness, and that approximately half the instances are satisfiable:

Set1 100 families of 5 instances, with $L = 10$, $N = 15$, $M = 10$, $Q = 1000$. The families have increasing K_k values from family 1 (capacities of about 1000) to family 100 (capacities of about 14000).

Set2 100 families of 5 instances, with $L = 10$, $N = 15$, $M = 10$, $Q = 60$. The families have increasing K_k values from family 1 (capacities of about 100) to family 100 (capacities of about 800).

Set3 20 families of 20 instances, with $L = 50$, $N = 15$, $M = 5$, $Q = 10$. The values of K_k increase in each family, ranging between 65 and 100.

All the instances have been encoded to SAT using the presented encodings for PB(AMO) constraints. The AMO Constraints (7.19) have been encoded with the well-known UP-maintaining GAC encoding referred as *regular* in [AM05] and *ladder* in [GN04], which only introduces a number of clauses and variables linear in the number of variables of the AMO constraint. Constraints (7.20) have been encoded with clauses $x_{i,1} \vee \cdots \vee x_{i,M}$, for all $1 \leq i \leq N$. We have used the Glucose 4.1 SAT solver [AS18] to solve the instances, on a 8GB, 3.10GHz Intel® Xeon® E3-1220v2.

The results are contained in Table 7.1. The evaluated encodings are the ones introduced in this paper, and their counterpart original ones. For completeness we also report results on the MDD-based Minimal Encoding from Chapter 4 (MDD), and its version without taking AMOs into account (BDD), from [ANO$^+$12]. For each encoding we report solving times on each dataset and the average size required to encode a PB(AMO) constraint. The size results do not include the number of variables and clauses introduced by AMO constraints (7.19), which is the same for all encodings and negligible.

In summary, it can be observed a dramatic decrease in size, and hence in generation time, as well a significant decrease in solving time, in all the generalized encodings for PB(AMO)s w.r.t. the original encodings for PBs. In most of cases the size and solving time reduction is of one order of magnitude. Even in Set3, which is the one with smallest AMO constraints (only 5 variables per AMO) the reduction is notable. The GPW encoding is the smallest, and the one which is less reduced when using the AMO constraints.

In Set1, which contains instances with large coefficients, the best approach is GGPW. Although this encoding do not UP-maintain GAC, the number of clauses and variables is remarkably small compared to the other encodings, whose size is proportional to K. In particular this dataset is prohibitive for GT and GGT encodings, which require a number of clauses quadratic in the value of K.

Set2 is similar to Set1 but it contains instances with small coefficients. In this case, the best approaches are MDD and GSWC, whose sizes are reasonably smaller than the ones in Set1. The GGT encoding introduces the largest number of clauses in this dataset, and has the worst time performance.

Instances in Set3 contain more PB constraints than the other datasets, and the values of K are distributed around the transition value from unsatisfiable instances to satisfiable instances. We have observed empirically that it is in this transition where the instances become harder. In this case, GGPW has the worst time performance although it still has a smaller size than the other encodings. This may be because GGPW is the only one which does not UP-maintain GAC.

	enc.	Q1	med	Q3	avg	t.o.	v.	cl.	g.t.
Set1	**BDD**	14.00	17.59	t.o.	219	158	857	1714	35.6
	SWC	10.51	14.12	t.o.	199	144	1100	2177	17.2
	GT	—	—	—	—	—	—	—	—
	GPW	0.93	0.97	23	114	85	5.9	77	0.8
	MDD	3.89	14.78	73	131	87	25	266	3.71
	GSWC	4.50	5.92	277	158	112	105	1076	10.01
	GGT	—	—	—	—	—	—	—	—
	GGPW	0.04	0.04	5.54	93	67	1.0	4.4	0.05
Set2	**BDD**	4.29	5.65	133	141	96	57	115	2.0
	SWC	4.10	5.41	138	140	95	68	135	1.3
	GT	5.33	6.94	182	154	110	10	1640	18.0
	GPW	0.46	0.48	11	108	77	3.5	42	0.4
	MDD	0.21	0.41	1.42	74	53	2.1	21	0.28
	GSWC	0.58	0.62	1.09	71	52	6.4	66	0.62
	GGT	2.42	8.83	53	132	95	1.9	120	1.53
	GGPW	0.02	0.03	3.36	89	65	0.6	2.5	0.03
Set3	**BDD**	215	t.o.	t.o.	423	218	4.8	9.6	0.7
	SWC	247	t.o.	t.o.	429	227	6.0	12	0.6
	GT	240	t.o.	t.o.	427	223	1.3	31	1.6
	GPW	172	t.o.	t.o.	415	229	0.8	5.1	0.3
	MDD	16.2	78.6	525	221	97	0.4	2.6	0.17
	GSWC	17.5	87.3	597	225	100	1.1	6.5	0.32
	GGT	70.8	281	t.o.	322	152	0.4	4.6	0.25
	GGPW	133	t.o.	t.o.	407	226	0.3	1.2	0.07

Table 7.1: From left to right: first quartile (Q1), median (med) and third quartile (Q3) of solving times (in seconds); average solving time in seconds (avg) counting time outs as 600 seconds; number of instances that timed out before being solved (t.o.); in thousands, average number of auxiliary variables (v.) and clauses (cl.) needed to encode one of the Constraints (7.18); in seconds, average time required to generate the CNF formula of an instance (g.t.). Solving time out (t.o.) is set to 600 seconds. Long dash (—) means that the resulting formulas are too large and their generation either run out of memory or did not finish in less than 600 seconds (in these instances we have been able to identify constraints requiring 33,000,000 clauses).

7.6 Chapter Summary

In this chapter we have provided different SAT encodings of PB(AMO)s, i.e., conjunctions of PB and AMO constraints. Namely, the new encodings are the Generalized Sequential Weight Counter encoding, the Generalized Generalized Totalizer encoding, and the Generalized Global Polynomial Watchdog encoding. These new encodings have been defined by generalising existing state-of-the-art SAT encodings of PB constraints, in a way that the size is highly reduced thanks to assuming that the AMO constraints are already enforced. Moreover, the propagation properties of the original encodings are preserved in the new ones. Our results show that all the new encodings are dramatically smaller and more efficient than their counterpart PB encodings. We have observed size reductions of an order of magnitude and solving time improvements of 1 or 2 orders of magnitude in many cases.

We have also shown that there is no best encoding for PB(AMO)s but it depends on the characteristics of the instances at hand. The datasets that we provide expose some strengths and weaknesses of the different encodings.

Chapter 8

Conclusions and Future Work

Scheduling problems are of substantial interest in the research community. On the one hand they have a myriad of practical application in industry and services. On the other hand, scheduling problems are usually hard, and contain challenging constraints that motivate the pursuit of efficient solving techniques. In this thesis we have provided new techniques for solving scheduling problems which are state-of-the-art among the exact solving approaches. Our techniques consist of using SMT to formulate scheduling problems for which we find solutions using an SMT solver as an oracle. We have also identified some opportunities to use collateral constraints to improve existing SAT encodings of PB constraints. Therefore, new compact encodings have been presented for them. Although these PB constraints play a central role in our scheduling formulations, the new encodings can have application to problems other than scheduling ones. Summing up, the different objectives that we had in this thesis have been accomplished. Now we summarise and present our conclusions on the different research outputs derived from this thesis.

Using BDDs to encode resource constraints

We have explored encoding PB constraints to SAT formulas as a way of dealing with resource constraints. Using the MRCPSP, we have reported that this option provides a high performance, with a remarkable improvement in comparison to using LIA expressions for such constraints. Here we can appreciate the high efficiency of modern SAT solvers, which contain very optimised implementations of Unit Propagation, powerful learning mechanisms and efficient search heuristics. These implementations combine with the good propagation properties of BDD-based encodings of PB constraints, leading to fast solving procedures.

Encode PB constraints taking into account collateral constraints

Whereas it is true that SAT encodings of PB constraints can be efficiently handled by SAT solvers, they risk leading to very large SAT formulas when the constraints involve a large number of variables. Therefore it is of critical importance to make the SAT representations of the constraints as small as possible. We have introduced PB modulo $\mathcal{C}$ constraints, PB($\mathcal{C}$) in short, as a way of denoting PB constraints that appear in conjunction with —either explicit or implicit— collateral constraints over the variables of the PB. We propose to deal with PB($\mathcal{C}$) in a new way, consisting on first encoding the set of constraints $\mathcal{C}$ if needed, and then encoding a relaxation of the PB constraint where it is assumed that the accompanying $\mathcal{C}$ constraints hold. This approach is highly interesting in settings where the set of constraints $\mathcal{C}$ are logically implied, since we can then obtain potentially much smaller encodings of PB constraints.

We have provided SAT encodings for the particular setting of PB(AMO) constraints, i.e. conjunctions of PB and AMO constraints. First of all we have focused on improving BDD-based encodings, and we have provided an encoding —the Minimal Encoding— for PB(AMO) constraints based on specialised MDDs. We show that encoding the PB constraints of PB(AMO) constraints using our Minimal Encoding can give much smaller formulas than using classical BDD-based encodings, with a reduction of two orders of magnitude in some cases. This size reduction comes with no penalisation on the propagation strength of the MDD-based encoding, since it UP-maintains GAC on the encoded PB(AMO) constraints, assuming that the AMO constraints are encoded using some UP-maintaining GAC encoding. Also, we propose methods to deal with other PB($\mathcal{C}$) constraints, in particular PB(EO) and PB(IC) constraints. Still related with PB(AMO) constraints, we provide a new normalisation procedure to express the PB constraint in a PB(AMO) with all coefficients positive and a less-than-or-equal operator, but maintaining any desired polarity on the variables of the PB constraint. This is essential because our encodings require the constraints to be normalised in this form. Moreover, thanks to this it is possible to use collateral AMO constraints between literals of the variables of the PB constraints with any polarity.

We have provided additional encodings of PB(AMO) constraints that are not based on decision diagrams. Namely, we provide the Generalized Sequential Weight Counter (GSWC) encoding, the Generalized Generalized Totalizer (GGT) encoding, and the Generalized Global Polynomial Watchdog (GGPW) encoding. All these encodings are interesting since they provide different size complexities, which let to chose the better one for the problem at hand. GSWC

and GGT UP-maintain GAC, while GGPW does not but can create small SAT formulas even for PB constraints with large coefficients. The potential benefit of using such encodings of PB(AMO) constraints instead of their counterpart version for PB constraints is made evident by our results in solving application-independent sets of constraints. Therefore, we expect that these encodings can have a high impact on many different applications.

SMT provides high expressivity as well as efficiency

Using SMT we have been able to provide intuitive and easily readable formulations for many scheduling problems with different peculiarities. Taking the RCPSP as a basis, we easily represent a schedule by an integer variable for each activity, denoting its start time. Then, using IDL expressions we can easily express precedence constraints as differences between start times. The other main component of the RCPSP, the resource constraints, are formulated following a Time-indexed approach. We introduce Boolean variables to indicate whether an activity is running at a particular time instant, and then we express the renewable resource constraints with one PB(AMO) constraint for each time instant and resource. After reasoning on the effect of the precedence relations over parallel resource consumption of the activities, we have been able to provide a quick procedure to find implicit AMO relations between the variables of resource constraints. By using minimum path covers on the extended precedence graph, we are able to find a small cardinality partition of the variables of the PB resource constraints. We have shown that this mechanism can be used as a basis to find AMO constraints in scheduling problems involving precedence relations, like the RCPSP.

We overcome the potential drawback of Time-indexed formulations, that is dealing with long scheduling horizons, by finding a good upper bound for the makespan with fast greedy algorithms. The makespan is represented as a start time of a dummy finishing activity, and we minimise the corresponding integer variable by increasingly constrained calls to a back-end solver, and keeping all the learning between calls.

We tackle many extensions of the RCPSP which present further formulating challenges, but again the high expressivity of SMT lets us obtain intuitive formulations. In the MRCPSP, the inclusion of many execution modes is dealt with Boolean variables representing whether an activity runs in a particular mode. This lets us make the time lag of precedence relations conditional on the selected execution mode. Similarly, in the MRCPSP/max we can generalise the precedence relation to define any time lag, either positive or negative and not necessarily equal to the duration of the predecessor activity. In the multi-

mode variants there are many possible resource consumption values for the same activity, depending on the selected execution mode. This fact introduces more variability in PB resource constraints, which translates to having more variables in those constraints than the ones we would have with single modes. Nevertheless, the use of our MDD-based encoding for PB(AMO) constraints mitigates the effect that this variability would have over the depth of classical BDD-based representations of the constraint, since the number of layers required to represent a resource constraint is the same as in the single-mode case of the same constraint. It is worth noting that the other PB(AMO) encodings that we have presented have similar benefits: in the GSWC encoding, the number of weight counters of a particular resource constraint remains the same as in the single-mode case; in the GGT encoding, the size of the binary tree is the same; in the GGPW encoding, the maximum number of entries per bucket is not increased. In the RCPSP/t, the Time approach naturally deals with the variability of resource availabilities during the project execution, and again PB(AMO) encodings let us mitigate the effect that the time-dependency of the resource requirements have over the encoding size. In the MSPSP there is the additional challenge of finding a resource assignment for each activity as well as a start time. We introduce several sets of variables with combined semantics that let us naturally express the resource constraints using cardinality constraints. We also import a set of cumulative cuts from an existing MILP formulation to define a powerful set of implied constraints. These constraints are PB(AMO) constraints, and therefore we can obtain compact SAT encodings.

Our systems to solve scheduling problems have proven to be extremely competitive. We make them available in the web page of our research group [LAP]. We have compared the performance of our systems with that of the exact solvers for each considered problem that, to the best of our knowledge, have recently reported the best results. We always run all the experiments on identical machines for a clean comparison. In all the problems considered in this thesis our solvers are the best approach for most datasets. SAT and SMT currently arise inspire much interest in the Artificial Intelligence community, and new solvers will be developed that will probably enhance the performance of our formulations. However it is still not very common to find the use of SAT and SMT in Operations Research. We think that our results, as well as the large number of fine works on problem solving with SAT and SMT that are being published, make evident the need to regard SAT and SMT as efficient approaches to take into account.

8.1 Future Work

One of the most relevant contributions of this thesis is that we do not simply provide ad-hoc solutions for specific problems, but we introduce new methods and encodings that can be exported not only to other scheduling problems but to CSPs of different domains. Therefore, we think that this thesis motivates interesting further research with high probability of being fruitful.

It would be very interesting to study how our formulations can be adapted to deal with other variations of scheduling problems. For instance, we could tackle problems dealing with different objective functions such as minimisation of resource consumption, minimisation of delay penalisations, or maximisation of profits. Some of these objectives may be pseudo-Boolean, and also there may be collateral AMO constraints. In these cases it would be interesting to consider the use of BDD-based encodings. Also, we think that SMT could be explored to tackle real-life scheduling problems, either to achieve exact solving in instances of reasonable complexity, or as an embedded component of a heuristic solving system.

We have focused on a Time-indexed approach, which traditionally has provided the best results in solving the existing benchmark sets of the problems we consider. It would be interesting to explore encodings based on other approach. For instance, the most used alternative to the Time approach is the Task approach, which checks for every pair of activities (A_i, A_j) whether A_j is running when A_i starts. Then, for every resource and every activity A_i it is imposed that the activities that are running when A_i starts do not exceed the resource availability. These constraints can be expressed as PB constraints. We have noticed that, similarly to what happens in the Time formulations, the precedence relations introduce AMO constraints over the variables of the PB constraints. Moreover the multi-mode and time-dependent requests extensions could also produce PB(AMO) constraints in the Task approach.

We also think that it is worthwhile to consider pure SAT formulations for scheduling problems, i.e. replace the start time integer variables and IDL precedence constraints by Boolean variables and clauses. It is interesting to explore in what situations using a specialised IDL solver is the best option, or on the contrary when a pure SAT solver can be more efficient.

It is also interesting to further investigate in the use PB($\mathcal{C}$) constraints. From our point of view, PB($\mathcal{C}$) can be seen as a new global constraint applicable to many domains, which is interesting by itself and it can be handled with different solving approaches. Many other cases could be interesting in some applications, for instance the use of collateral cardinality constraints different than AMO and EO. At first sight do not seem obvious how to generalise

the PB(AMO) SAT encodings that we have presented to deal with collateral cardinality constraints, and it might be necessary to design novel fine-grained encodings.

We have focused on the compact encoding of PB constraints and therefore we tackle PB($\mathcal{C}$) constraints. However, we think that many other challenging constraints could be handled by taking into account collateral constraints. This could lead to new efficient encodings or specific propagators for different kinds of constraints.

Regarding PB(AMO) constraints, we have used them in the domain of scheduling, where we have designed a procedure to efficiently detect large AMO constraints. The detection of a good set of AMO constraints is the key point to get small SAT formulas. Some work could be done to automatically detect AMO constraints not only in scheduling problems but also in CSPs in general. Here, it is interesting to see if automatic methods could be better than the path cover method we have provided, and also to what extent the PB(AMO) constraints are useful in other domains.

Many of the exact systems that we compare with implement specific solving procedures for scheduling, like branching heuristics or global constraint propagators. However, our systems use the SMT solver as a black box which implements application-independent solving procedures. It is interesting to explore the integration of particular techniques for scheduling problems in the solving mechanisms of the SMT solver, such as specialised branching heuristics. Analysing the trace of the solver when solving scheduling problems could shed some light on the low-level reasons of the efficiency of SMT solvers. With a better understanding of the actual solving process, we could detect new improvement opportunities.

Bibliography

[AB93] Abderrahmane Aggoun and Nicolas Beldiceanu. Extending CHIP in Order to Solve Complex Scheduling and Placement Problems. *Mathematical and Computer Modelling*, 17(7):57–73, 1993.

[ABL13] Carlos Ansótegui, Maria Luisa Bonet, and Jordi Levy. SAT-Based MaxSAT Algorithms. *Artificial Intelligence*, 196:77–105, 2013.

[ABP+11] Carlos Ansótegui, Miquel Bofill, Miquel Palahí, Josep Suy, and Mateu Villaret. Satisfiability Modulo Theories: An Efficient Approach for the Resource-Constrained Project Scheduling Problem. In *Proceedings of the Ninth Symposium on Abstraction, Reformulation, and Approximation (SARA)*, pages 2–9, 2011.

[ACSdG16] Bernardo F Almeida, Isabel Correia, and Francisco Saldanha-da Gama. Priority-Based Heuristics for the Multi-Skill Resource Constrained Project Scheduling Problem. *Expert Systems with Applications*, 57:91–103, 2016.

[ADN13] Christian Artigues, Sophie Demassey, and Emmanuel Neron. *Resource-Constrained Project Scheduling: Models, Algorithms, Extensions and Applications*. John Wiley & Sons, 2013.

[AGMES16] Ignasi Abío, Graeme Gange, Valentin Mayer-Eichberger, and Peter J Stuckey. On CNF Encodings of Decision Diagrams. In *Proceedings of the 13th International Conference on Integration of AI and OR Techniques in Constraint Programming (CPAIOR)*, volume 9676 of *LNCS*, pages 1–17, 2016.

[AM05] Carlos Ansótegui and Felip Manyà. Mapping Problems with Finite-Domain Variables to Problems with Boolean Variables. In

Proceedings of the 7th International Conference on Theory and Applications of Satisfiability Testing (SAT) International Conference, volume 3542 of *LNCS*, pages 1–15. Springer, 2005.

[AMES15] Ignasi Abío, Valentin Mayer-Eichberger, and Peter J Stuckey. Encoding Linear Constraints with Implication Chains to CNF. In *Proceedings of the 21st International Conference on Principles and Practice of Constraint Programming (CP)*, volume 9255 of *LNCS*, pages 3–11. Springer, 2015.

[ANO+12] Ignasi Abío, Robert Nieuwenhuis, Albert Oliveras, Enric Rodríguez-Carbonell, and Valentin Mayer-Eichberger. A New Look at BDDs for Pseudo-Boolean Constraints. *Journal of Artificial Intelligence Research*, 45:443–480, 2012.

[ANOR11] Roberto Asín, Robert Nieuwenhuis, Albert Oliveras, and Enric Rodríguez-Carbonell. Cardinality Networks: a Theoretical and Empirical Study. *Constraints*, 16(2):195–221, 2011.

[ANORC13] Ignasi Abío, Robert Nieuwenhuis, Albert Oliveras, and Enric Rodríguez-Carbonell. A Parametric Approach for Smaller and Better Encodings of Cardinality Constraints. In *Proceedings of the 9th International Conference on Principles and Practice of Constraint Programming (CP)*, pages 80–96. Springer, 2013.

[Art13] Christian Artigues. A Note on Time-Indexed Formulations for the Resource-Constrained Project Scheduling Problem. 2013.

[AS14] Ignasi Abío and Peter J Stuckey. Encoding Llnear Constraints into SAT. In *Proceedings of the 20th International Conference on Principles and Practice of Constraint Programming (CP)*, volume 8656 of *LNCS*, pages 75–91, 2014.

[AS18] Gilles Audemard and Laurent Simon. On the Glucose SAT Solver. *International Journal on Artificial Intelligence Tools*, 27(1):1–25, 2018.

[BB03] Olivier Bailleux and Yacine Boufkhad. Efficient CNF Encoding of Boolean Cardinality Constraints. In *Proceedings of the 9th International Conference on Principles and Practice of Constraint Programming (CP)*, volume 2833 of *LNCS*, pages 108–122, 2003.

[BBMV13] Miquel Bofill, Dídac Busquets, Víctor Muñoz, and Mateu Villaret. Reformulation Based MaxSAT Robustness. *Constraints*, 18(2):202–235, 2013.

[BBR06] Olivier Bailleux, Yacine Boufkhad, and Olivier Roussel. A Translation of Pseudo-Boolean Constraints to SAT. *Journal on Satisfiability, Boolean Modeling and Computation*, 2:191–200, 2006.

[BBR09] Olivier Bailleux, Yacine Boufkhad, and Olivier Roussel. New Encodings of Pseudo-Boolean Constraints into CNF. In *Proceedings of the 12th International Conference on Theory and Applications of Satisfiability Testing (SAT)*, volume 5584 of *LNCS*, pages 181–194. Springer, 2009.

[BCF+06] Roberto Bruttomesso, Alessandro Cimatti, Anders Franzén, Alberto Griggio, Alessandro Santuari, and Roberto Sebastiani. To Ackermann-ize or Not to Ackermann-ize? On Efficiently Handling Uninterpreted Function Symbols in $SMT(EUF \cup T)$. In *Proceedings of the 13th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR)*, pages 557–571, 2006.

[BCSV16] Miquel Bofill, Jordi Coll, Josep Suy, and Mateu Villaret. Solving the Multi-Mode Resource-Constrained Project Scheduling Problem with SMT. In *Proceedings of the 28th IEEE International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 239–246, 2016.

[BCSV17a] Miquel Bofill, Jordi Coll, Josep Suy, and Mateu Villaret. An Efficient SMT Approach to Solve MRCPSP/max Instances with Tight Constraints on Resources. In *Proceedings of the 23rd International Conference on Principles and Practice of Constraint Programming (CP)*, volume 10416 of *LNCS*, pages 71–79. Springer, 2017.

[BCSV17b] Miquel Bofill, Jordi Coll, Josep Suy, and Mateu Villaret. Compact MDDs for Pseudo-Boolean Constraints with At-Most-One Relations in Resource-Constrained Scheduling Problems. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 555–562, 2017.

[BCSV19] Miquel Bofill, Jordi Coll, Josep Suy, and Mateu Villaret. SAT Encodings of Pseudo-Boolean Constraints with At-Most-One

Relations. In *Proceedings of the 16th International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research (CPAIOR) (To appear)*, 2019.

[BDM+99] Peter Brucker, Andreas Drexl, Rolf Möhring, Klaus Neumann, and Erwin Pesch. Resource-Constrained Project Scheduling: Notation, Classification, Models, and Methods. *European Journal of Operational Research*, 112(1):3 – 41, 1999.

[BHvMW09] Armin Biere, Marijn J. H. Heule, Hans van Maaren, and Toby Walsh, editors. *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, February 2009.

[BLK83a] J. Blazewicz, J. K. Lenstra, and A. Rinnooy Kan. Scheduling Subject to Resource Constraints: Classification and Complexity. *Discrete Applied Mathematics*, 5(1):11 – 24, 1983.

[BLK83b] Jacek Blazewicz, Jan Karel Lenstra, and AHG Rinnooy Kan. Scheduling Subject to Resource Constraints: Classification and Complexity. *Discrete Applied Mathematics*, 5(1):11–24, 1983.

[BM08] Odile Bellenguez-Morineau. *Methods to Solve Multi-Skill Project Scheduling Problem*. PhD thesis, Université François-Rabelais de Tours, 2008.

[BPSV12] Miquel Bofill, Miquel Palahí, Josep Suy, and Mateu Villaret. Solving Constraint Satisfaction Problems with SAT Modulo Theories. *Constraints*, 17(3):273–303, 2012.

[BPSV14] Miquel Bofill, Miquel Palahí, Josep Suy, and Mateu Villaret. Solving Intensional Weighted CSPs by Incremental Optimization with BDDs. In *Proceedings of the 20th International Conference on Principles and Practice of Constraint Programming (CP)*, pages 207–223. Springer, 2014.

[Bry86] Randal E Bryant. Graph-Based Algorithms for Boolean Function Manipulation. *IEEE Transactions on Computers*, C-35(8):677–691, 1986.

[BST10] Clark Barrett, Aaron Stump, and Cesare Tinelli. The Satisfiability Modulo Theories Library (SMT-LIB). `www.SMT-LIB.org`, 2010.

[BW03] Fahiem Bacchus and Jonathan Winter. Effective Preprocessing with Hyper-Resolution and Equality Reduction. In *Proceedings of the 6th International Conference on Theory and Applications of Satisfiability Testing (SAT)*, volume 2919 of *LNCS*, pages 341–355, 2003.

[Cas12] Carlos Eduardo Montoya Casas. *New Methods for the Multi-Skills Project Scheduling Problem.* PhD thesis, Ecole des Mines de Nantes, 2012.

[Chu11] Geoffrey G. Chu. *Improving Combinatorial Optimization.* PhD thesis, The University of Melbourne, 2011.

[CLSdG12] Isabel Correia, Lídia Lampreia Lourenço, and Francisco Saldanha-da Gama. Project Scheduling with Flexible Resources: Formulation and Inequalities. *OR Spectrum*, 34(3):635–663, 2012.

[Coo71] Stephen A Cook. The Complexity of Theorem-Proving Procedures. In *Proceedings of the 3rd annual ACM symposium on Theory of computing*, pages 151–158. ACM, 1971.

[CSE14] Ripon Kumar Chakrabortty, Ruhul Amin Sarker, and Daryl L Essam. Event Based Approaches for Solving Multi-Mode Resource Constraints Project Scheduling Problem. In *Computer Information Systems and Industrial Management*, pages 375–386. Springer, 2014.

[CV11] José Coelho and Mario Vanhoucke. Multi-Mode Resource-Constrained Project Scheduling Using RCPSP and SAT Solvers. *European Journal of Operational Research*, 213(1):73–82, 2011.

[Dav73] Edward W Davis. Project Scheduling Under Resource Constraints —Historical Review and Categorization of Procedures. *AIIE Transactions*, 5(4):297–313, 1973.

[DdM06a] B. Dutertre and L. de Moura. The Yices SMT Solver. Technical report, Computer Science Laboratory, SRI International, 2006. Available at `http://yices.csl.sri.com`.

[DdM06b] Bruno Dutertre and Leonardo de Moura. Integrating Simplex with DPLL(T). Technical Report SRI-CSL-06-01, Computer Science Laboratory, SRI International, May 2006. `http://yices.csl.sri.com/papers/sri-csl-06-01.pdf`.

[DLL62] Martin Davis, George Logemann, and Donald W. Loveland. A Machine Program for Theorem-Proving. *Communications of the ACM*, 5(7):394–397, 1962.

[dMB08] Leonardo Mendonça de Moura and Nikolaj Bjørner. Z3: An Efficient SMT Solver. In *Proceedings of the 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 4963 of *LNCS*, pages 337–340. Springer, 2008.

[dMR04] L. de Moura and H. Ruess. An Experimental Evaluation of Ground Decision Procedures. In *Proceedings of the 16th International Conference Computer Aided Verification (CAV)*, volume 3114 of *LNCS*, pages 162–174, 2004.

[DP60] Martin Davis and Hilary Putnam. A Computing Procedure for Quantification Theory. *Journal of the ACM*, 7(3):201–215, 1960.

[DR59] George B Dantzig and John H Ramser. The Truck Dispatching Problem. *Management Science*, 6(1):80–91, 1959.

[DRH99] Bert De Reyck and Willy Herroelen. The Multi-Mode Resource-Constrained Project Scheduling Problem with Generalized Precedence Relations. *European Journal of Operational Research*, 119(2):538–556, 1999.

[Dut14] Bruno Dutertre. Yices 2.2. In *Computer-Aided Verification (CAV'2014)*, volume 8559 of *LNCS*, pages 737–744. Springer, July 2014.

[DVV03] Sven De Vries and Rakesh V Vohra. Combinatorial Auctions: a Survey. *INFORMS Journal on Computing*, 15(3):284–309, 2003.

[EB05] Niklas Eén and Armin Biere. Effective Preprocessing in SAT Through Variable and Clause Elimination. In *Proceedings of the 8th International Conference on Theory and Applications of Satisfiability Testing (SAT)*, volume 3569 of *LNCS*, pages 61–75, 2005.

[ES06] Niklas Eén and Niklas Sorensson. Translating pseudo-Boolean Constraints into SAT. *Journal on Satisfiability, Boolean Modeling and Computation*, 2:1–26, 2006.

[FS09] Thibaut Feydy and Peter J Stuckey. Lazy clause generation reengineered. In *Proceedings of the 15th International Conference on Principles and Practice of Constraint Programming (CP)*, pages 352–366. Springer, 2009.

[Gei13] Martin Josef Geiger. Iterated Variable Neighborhood Search for the Resource Constrained Multi-Mode Multi-Project Scheduling Problem. *arXiv preprint arXiv:1310.0602*, 2013.

[GJ75] Michael R Garey and David S. Johnson. Complexity Results for Multiprocessor Scheduling Under Resource Constraints. *SIAM Journal on Computing*, 4(4):397–411, 1975.

[GN04] Ian P Gent and Peter Nightingale. A New Encoding of Alldifferent into SAT. In *3rd International Workshop on Modelling and Reformulating Constraint Satisfaction (ModRef)*, pages 95–110, 2004.

[Har99] Sönke Hartmann. *Project Scheduling Under Limited Resources: Models, Methods, and Applications*, volume 478. Springer Science & Business Media, 1999.

[Har13] Sönke Hartmann. Project Scheduling with Resource Capacities and Requests Varying with Time: a Case Study. *Flexible Services and Manufacturing Journal*, 25(1-2):74–93, 2013.

[Har15] Sönke Hartmann. Time-Varying Resource Requirements and Capacities. In *Handbook on Project Management and Scheduling Vol. 1*, pages 163–176. Springer, 2015.

[HB10] Sönke Hartmann and Dirk Briskorn. A Survey of Variants and Extensions of the Resource-Constrained Project Scheduling Problem. *European Journal of Operational Research*, 207(1):1–14, 2010.

[HDR99] Willy Herroelen, Erik Demeulemeester, and Bert Reyck. A Classification Scheme for Project Scheduling. In *Project Scheduling*, volume 14 of *International Series in Operations Research & Management Science*, pages 1–26. Springer US, 1999.

[HK73] John E Hopcroft and Richard M Karp. An nˆ5/2 Algorithm for Maximum Matchings in Bipartite Graphs. *SIAM Journal on Computing*, 2(4):225–231, 1973.

[HLS10] Bing Han, Jimmy Leblet, and Gwendal Simon. Hard Multidimensional Multiple Choice Knapsack Problems, an Empirical Study. *Computers & Operations Research*, 37(1):172 – 181, 2010.

[HMS12] Steffen Hölldobler, Norbert Manthey, and Peter Steinke. A Compact Encoding of Pseudo-Boolean Constraints into SAT. In *KI 2012: Advances in Artificial Intelligence - 35th Annual German Conference on Artificial Intelligence*, volume 7526 of *LNCS*, pages 107–118. Springer, 2012.

[HTY94] Kazuhisa Hosaka, Yasuhiko Takenaga, and Shuzo Yajima. On the Size of Ordered Binary Decision Diagrams Representing Threshold Functions. In *Proceedings of the 5th International Symposium on Algorithms and Computation (ISAAC)*, volume 834 of *LNCS*, pages 584–592, 1994.

[ISEZ93] Oya Icmeli, S Selcuk Erenguc, and Christopher J Zappe. Project Scheduling Problems: a Survey. *International Journal of Operations & Production Management*, 13(11):80–91, 1993.

[JMM15] Saurabh Joshi, Ruben Martins, and Vasco M. Manquinho. Generalized Totalizer Encoding for Pseudo-Boolean Constraints. In *Proceedings of the 21st International Conference on Principles and Practice of Constraint Programming (CP)*, volume 9255 of *LNCS*, pages 200–209. Springer, 2015.

[KALM11] Oumar Koné, Christian Artigues, Pierre Lopez, and Marcel Mongeau. Event-Based MILP Models for Resource-Constrained Project Scheduling Problems. *Computers & Operations Research*, 38:3–13, January 2011.

[Kel63] James E Kelley. The critical-path method: Resources planning and scheduling. *Industrial scheduling*, 13:347–365, 1963.

[KKG12] Thomas S. Kyriakidis, Georgios M. Kopanos, and Michael C. Georgiadis. MILP formulations for single- and multi-mode resource-constrained project scheduling problems. *Computers & Chemical Engineering*, 36(0):369 – 385, 2012.

[Kol96] Rainer Kolisch. Serial and parallel resource-constrained project scheduling methods revisited: Theory and computation. *European Journal of Operational Research*, 90(2):320–333, 1996.

[KS97] Rainer Kolisch and Arno Sprecher. PSPLIB - A Project Scheduling Problem Library. *European Journal of Operational Research*, 96(1):205–216, 1997.

[KSD95] Rainer Kolisch, Arno Sprecher, and Andreas Drexl. Characterization and Generation of a General Class of Resource-Constrained Project Scheduling Problems. *Management science*, 41(10):1693–1703, 1995.

[LAP] Logic and Programming group web page. `http://ima.udg.edu/Recerca/lap/`. Accessed: April 2019.

[Lap92] Gilbert Laporte. The Vehicle Routing Problem: An Overview of Exact and Approximate Algorithms. *European Journal of Operational Research*, 59(3):345–358, 1992.

[LM09] Chu Min Li and Felip Manyà. MaxSAT, Hard and Soft Constraints. In *Handbook of Satisfiability*, pages 613–631. 2009.

[MBMPR14] Carlos Montoya, Odile Bellenguez-Morineau, Eric Pinson, and David Rivreau. Branch-and-Price Approach for the Multi-Skill Project Scheduling Problem. *Optimization Letters*, 8(5):1721–1734, 2014.

[MMZ+01] Matthew W Moskewicz, Conor F Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: Engineering an efficient sat solver. In *Proceedings of the 38th annual Design Automation Conference*, pages 530–535. ACM, 2001.

[MPS14] Norbert Manthey, Tobias Philipp, and Peter Steinke. A More Compact Translation of Pseudo-Boolean Constraints into CNF Such That Generalized Arc Consistency Is Maintained. In *KI 2014: Advances in Artificial Intelligence - 37th Annual German Conference on AI*, volume 8736 of *LNCS*, pages 123–134, 2014.

[MSS99] João P Marques-Silva and Karem A Sakallah. GRASP: A Search Algorithm for Propositional Satisfiability. *IEEE Transactions on Computers*, 48(5):506–521, 1999.

[MTZ60] Clair E Miller, Albert W Tucker, and Richard A Zemlin. Integer programming formulation of traveling salesman problems. *Journal of the ACM*, 7(4):326–329, 1960.

[NH79] Simeon C. Ntafos and S. Louis Hakimi. On path cover problems in digraphs and applications to program testing. *IEEE Transactions on Software Engineering*, (5):520–529, 1979.

[PH74] James H Patterson and Walter D Huber. A Horizon-Varying, Zero-One Approach to Project Scheduling. *Management Science*, 20(6):990–998, 1974.

[PS15] Tobias Philipp and Peter Steinke. PBLib - A Library for Encoding Pseudo-Boolean Constraints into CNF. In *Proceedings of the 18th International Conference on Theory and Applications of Satisfiability Testing (SAT)*, volume 9340 of *LNCS*, pages 9–16. Springer, 2015.

[PW96] Lawrence J. Watters Pritsker, A. Alan B. and Philip S. Wolfe. Multiproject Scheduling with Limited Resources: A Zero-One Programming Approach. *Management Science*, 16:93–108, 1996.

[RVBW06] Francesca Rossi, Peter Van Beek, and Toby Walsh. *Handbook of Constraint Programming*. Elsevier, 2006.

[Sch98] Christoph Schwindt. *Generation of Resource Constrained Project Scheduling Problems Subject to Temporal Constraints*. Inst. für Wirtschaftstheorie und Operations-Research, 1998.

[SD98] Arno Sprecher and Andreas Drexl. Multi-Mode Resource-Constrained Project Scheduling by a Simple, General and Powerful Sequencing Slgorithm. *European Journal of Operational Research*, 107(2):431 – 450, 1998.

[SDK78] Joel P Stinson, Edward W Davis, and Basheer M Khumawala. Multiple Resource-Constrained Scheduling Using Branch and Bound. *AIIE Transactions*, 10(3):252–259, 1978.

[SdlBM+05] Peter J Stuckey, Maria Garcia de la Banda, Michael Maher, Kim Marriott, John Slaney, Zoltan Somogyi, Mark Wallace, and Toby Walsh. The G12 Project: Mapping Solver Independent Models to Efficient Solutions. In *International Conference on Logic Programming*, pages 9–13. Springer, 2005.

[Seb07] Roberto Sebastiani. Lazy Satisability Modulo Theories. *Journal on Satisfiability, Boolean Modeling and Computation*, 3(3-4):141–224, 2007.

[SFS13] Andreas Schutt, Thibaut Feydy, and Peter J Stuckey. Explaining Time-Table-Edge-Finding Propagation for the Cumulative Resource Constraint. In *Proceedings of the 10th International Conference on AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR)*, pages 234–250. Springer, 2013.

[SFSW09] Andreas Schutt, Thibaut Feydy, Peter J. Stuckey, and Mark Wallace. Why Cumulative Decomposition Is Not as Bad as It Sounds. In *Proceedings of the 15th International Conference on Principles and Practice of Constraint Programming (CP)*, volume 5732 of *LNCS*, pages 746–761. Springer, 2009.

[SFSW13] Andreas Schutt, Thibaut Feydy, Peter J Stuckey, and Mark G Wallace. Solving RCPSP/max by Lazy Clause Generation. *Journal of Scheduling*, 16(3):273–289, 2013.

[SH16] Alexander Schnell and Richard F Hartl. On the Efficient Modeling and Solution of the Multi-Mode Resource-Constrained Project Scheduling Problem with Generalized Precedence Relations. *OR Spectrum*, 38(2):283–303, 2016.

[SHMB90] Arvind Srinivasan, Timothy Ham, Sharad Malik, and Robert K Brayton. Algorithms for discrete function manipulation. In *IEEE International Conference on Computer-Aided Design (ICCAD)*, pages 92–95. IEEE, 1990.

[SS77] Richard M. Stallman and Gerald J. Sussman. Forward Reasoning and Dependency-Directed Backtracking in a System for Computer-Aided Circuit Analysis. *Artificial Intelligence*, 9(2):135–196, 1977.

[SS16] Ria Szeredi and Andreas Schutt. Modelling and Solving Multi-Mode Resource-Constrained Project Scheduling. In *Proceedings of the 22nd International Conference on Principles and Practice of Constraint Programming (CP)*, pages 483–492. Springer, 2016.

[ST12] Roberto Sebastiani and Silvia Tomasi. Optimization in SMT with $\mathcal{LA}(\mathbb{Q})$ Cost Functions. In *Proceedings of the 6th International Joint Conference on Automated Reasoning (IJCAR)*, volume 7364 of *LNCS*, pages 484–498. Springer, 2012.

[ST15] Roberto Sebastiani and Patrick Trentin. OptiMathSAT: A Tool for Optimization Modulo Theories. In *Proceedings of the 27th International Conference on Computer Aided Verification (CAV)*, pages 447–454. Springer, 2015.

[Suy13] Josep Suy. *A Satisfiability Modulo Theories Approach to Constraint Programming.* PhD thesis, Universitat de Girona, 2013.

[TBS13] Naoyuki Tamura, Mutsunori Banbara, and Takehide Soh. Compiling Pseudo-Boolean Constraints to SAT with Order Encoding. In *25th IEEE International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 1020–1027, 2013.

[TP78] F Brian Talbot and James H Patterson. An Efficient Integer Programming Algorithm with Network Cuts for Solving Resource-Constrained Scheduling Problems. *Management Science*, 24(11):1163–1174, 1978.

[TSCS16] Túlio AM Toffolo, Haroldo G Santos, Marco AM Carvalho, and Janniele A Soares. An Integer Programming Approach to the Multimode Resource-Constrained Multiproject Scheduling Problem. *Journal of Scheduling*, 19(3):295–307, 2016.

[Vil11] Petr Vilím. Timetable Edge Finding Filtering Algorithm for Discrete Cumulative Resources. In *Proceedings of the 18th International Conference on AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR)*, pages 230–245. Springer, 2011.

[VLS15] Petr Vilím, Philippe Laborie, and Paul Shaw. Failure-Directed Search for Constraint-Based Scheduling. In *Proceedings of the 12th Integration of AI and OR Techniques in Constraint Programming (CPAIOR)*, pages 437–453. Springer, 2015.

[VPV14] Vincent Van Peteghem and Mario Vanhoucke. An Experimental Investigation of Metaheuristics for the Multi-Mode Resource-Constrained Project Scheduling Problem on New Dataset Instances. *European Journal of Operational Research*, 235(1):62–72, 2014.

[YFS17] Kenneth D Young, Thibaut Feydy, and Andreas Schutt. Constraint Programming Applied to the Multi-Skill Project Scheduling Problem. In *Proceedings of the 23rd International Conference*

on Principles and Practice of Constraint Programming (CP), pages 308–317. Springer, 2017.

[ZBY06] Guidong Zhu, Jonathan F. Bard, and Gang Yu. A Branch-and-Cut Procedure for the Multimode Resource-Constrained Project-Scheduling Problem. *INFORMS J. on Computing*, 18(3):377–390, January 2006.

[ZHR08] Juan Camilo Zapata, Bri Mathias Hodge, and Gintaras V. Reklaitis. The Multimode Resource Constrained Multiproject Scheduling Problem: Alternative Formulations. *AIChE Journal*, 54(8):2101–2119, 2008.

[ZM02] Lintao Zhang and Sharad Malik. The Quest for Efficient Boolean Satisfiability Solvers. In *Proceedings of the 14th International Conference on Computer Aided Verification (CAV)*, volume 2404 of *LNCS*, pages 17–36. Springer, 2002.

Alphabetical Index