NNT : 2017SACLX122

Thèse de doctorat de l'Université Paris-Saclay préparée à l'École Polytechnique

École Doctorale n°580
Sciences et Technologies de l'Information et de la Communication

Mathématiques et Informatique

Par

M. Nicolas Bonifas

Geometric and Dual Approaches to Cumulative Scheduling

Thèse présentée et soutenue à Paris, le 19 décembre 2017.

Composition du Jury :
M. Jean-Charles Billaut, Professeur, Université de Tours. Président
M. Jacques Carlier, Professeur Émérite, UT Compiègne. Rapporteur
M. Christian Artigues, Directeur de Recherche, LAAS-CNRS. Rapporteur
Mme Catuscia Palamidessi, Directrice de Recherche, LIX-CNRS. Examinatrice
M. Peter Stuckey, Professeur, Université de Melbourne. Examinateur
M. Philippe Baptiste, Directeur de Recherche, LIX-CNRS. Directeur de thèse
M. Jérôme Rogerie, Ingénieur R&D, IBM France. Co-directeur de thèse

Abstract

Context

This work falls in the scope of mathematical optimization, and more precisely in constraint-based scheduling. This involves determining the start and end dates for the execution of tasks (these dates constitute the decision variables of the optimization problem), while satisfying time and resource constraints and optimizing an objective function.

In constraint programming, a problem of this nature is solved by a tree-based exploration of the domains of decision variables with a branch & bound search. In addition, at each node a propagation phase is carried out: necessary conditions for the different constraints to be satisfied are verified, and values of the domains of the decision variables which do not satisfy these conditions are eliminated.

In this framework, the most frequently encountered resource constraint is the cumulative. Indeed, it enables modeling parallel processes, which consume during their execution a shared resource available in finite quantity (such as machines or a budget). Propagating this constraint efficiently is therefore of crucial importance for the practical efficiency of a constraint-based scheduling engine.

In this thesis, we study the cumulative constraint with the help of tools rarely used in constraint programming (polyhedral analysis, linear programming duality, projective geometry duality). Using these tools, we propose two contributions for the domain: the Cumulative Strengthening, and the $O(n^2 \log n)$ Energy Reasoning.

Contributions

Cumulative Strengthening

We propose a reformulation of the cumulative constraint, that is to say the generation of tighter redundant constraints which allows for a stronger propa-

gation, of course without losing feasible solutions.

This technique is commonly used in integer linear programming (cutting planes generation), but this is one of its very first examples of a redundant global constraint in constraint programming.

Calculating this reformulation is based on a linear program whose size depends only on the capacity of the resource but not on the number of tasks, which makes it possible to precompute the reformulations.

We also provide guarantees on the quality of the reformulations thus obtained, showing in particular that all bounds calculated using these reformulations are at least as strong as those that would be obtained by a preemptive relaxation of the scheduling problem.

This technique makes it possible to strengthen all propagations of the cumulative constraint based on the calculation of an energy bound, in particular the Edge-Finding and the Energy Reasoning.

This work was presented at the ROADEF 2014 conference [BB14] and was published in 2017 in the Discrete Applied Mathematics journal [BB17a].

$O(n^2 \log n)$ Energy Reasoning

This work consists of an improvement of the algorithmic complexity of one of the most powerful propagations for the cumulative constraint, namely the Energy Reasoning, introduced in 1990 by Erschler and Lopez.

In spite of the very strong deductions found by this propagation, it is rarely used in practice because of its cubic complexity. Many approaches have been developed in recent years to attempt to make it usable in practice (machine learning to use it wisely, reducing the constant factor of its algorithmic complexity, etc).

We propose an algorithm that computes this propagation with a $O(n^2 \log n)$ complexity, which constitutes a significant improvement of this algorithm known for more than 25 years. This new approach relies on new properties of the cumulative constraint and on a geometric study.

This work was published in preliminary form at the CP 2014 conference [Bon14] and was published [Bon16] at the ROADEF 2016 conference, where it was awarded 2$^{\text{th}}$ prize of the Young Researcher Award.

Remerciements

Preface

Contents

Chapter 1

Introduction

Scheduling consists in planning the realization of a project, defined as a set of tasks, by setting execution dates for each of the tasks and allocating scarce resources to the tasks over time. In this introductory chapter, we begin by briefly recalling the history of scheduling and its emergence as a topic of major interest in Computer Science in Section 1.1. In Section 1.2 we underline the essential contribution of mathematical optimization to scheduling. We conclude the chapter by presenting in Section 1.3 our main contributions and an outline of the thesis.

1.1 The birth of Scheduling within Computer Science

Scheduling as a discipline first appeared in the field of work management in the USA during WW1 with the invention of the Gantt chart. Its creator, Henry Gantt, was an American engineer and management consultant who worked with Frederick Taylor on the so-called scientific management.

This method was enhanced with the PERT tool in 1955, which was developed as a collaboration between the US Navy, Lockheed and Booz Allen Hamilton. Similar tools were developed independently around the same time, for example CPM (Critical Path Method) at DuPont and MPM (Méthode des Potentiels Métra) by Bernard Roy in France, both in 1958. These tools proved invaluable in managing complex projects of the time, such as the Apollo program. They are still widely used today in project planning thanks to their simplicity, which also imposes severe limitations on the accuracy of the models which can be expressed with these tools. In modern terms, these techniques embed a precedence network only, with no resource constraints. Through operations re-

search think tanks and academic exchanges, many prominent mathematicians and early computer scientists came in contact with these questions.

At the same time in the early 1960s, with the advent of parallel computers and time sharing on mainframes, scheduling jobs on computers became a topic of major interest for operating systems designers, and the problem emerged as its own field of research in Computer Science. This is all the more the case since scheduling, perhaps coincidentally, involves very fundamental objects in Computer Science, and was the source of many examples in the early development of graph theory, complexity theory and approximation theory.

The widespread, strategic applications of scheduling today justify an ever-increased research effort. To mention a few application areas, scheduling is used in organizing the production steps in workshops, both manned and automated. It is used in building rosters and timetables in complex organizations, such as hospitals and airlines. In scientific computing, it can be used to schedule computations on clusters of heterogenous machines [ABED+15]. Recently, in a beautiful application, the sequence of scientific operations performed by the Philae lander on comet 67P/Churyumov–Gerasimenko (see Figure 1.1) was planned using constraint-based scheduling in a context of severely limited energetic resources, processing power and communication bandwidth [SAHL12]. Close to half of the optimization problems submitted to IBM optimization consultants have a major scheduling element.

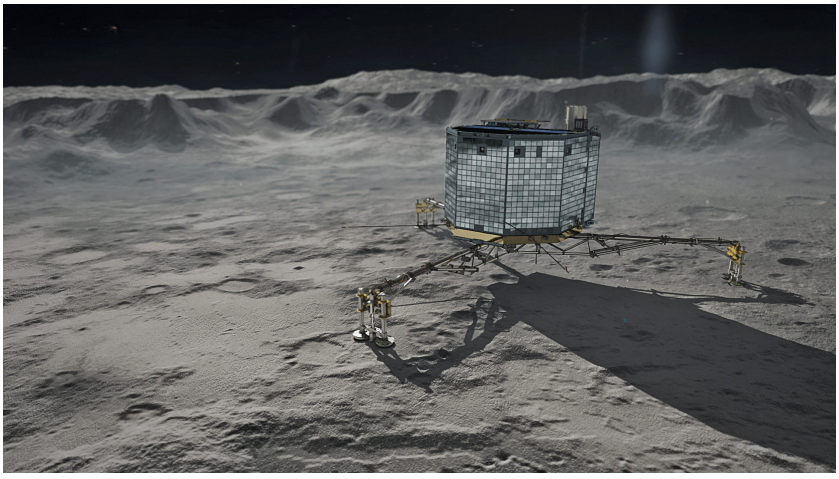


Figure 1.1: Depiction of Philae on comet 67P/C-G (Credits: Deutsches Zentrum für Luft- und Raumfahrt, CC-BY 3.0)

1.2 The Optimization approach to Scheduling

Constraint-based scheduling finds its origins in three scientific domains: Operations Research, Constraint Programming and Mathematical Optimization. We describe the respective contributions of these fields to the state of the art of scheduling engines, and give an overview of the modeling principles when using a modern, Model & Run, solver.

1.2.1 Operations Research

Starting in the early 1960s, several specific scheduling problems have been extensively studied in the Operations Research and the Algorithms communities.

There were originally two distinct lines of research, each with their own technical results and applications: one of them is *machine scheduling*, which studies problems such as *parallel machine scheduling* and *shop scheduling* (these notions will be defined later in section 2.1), for example to plan tasks on the different machines in a factory, and the other is *project management*, where the *RCPSP* (defined later in section 3.2) is used to plan the tasks of a project in a large organization.

The models which were produced are simple, restricted in expressivity, and each of them relies on an ad-hoc algorithm. In spite of the success of this approach when it is focused on a specific, simplified problem, the solutions are not robust to the addition of most side constraints.

These models must reflect the data structures of the underlying algorithms, and thus often lack expressiveness. Not taking into account side constraints results in the production of simplified solutions, which are sometimes difficult to translate back to the original problem, or can be very far from optimal solutions to the complete problem. This limitation significantly restricts the applicability of this approach to practical problems. Moreover, the solving algorithms lack genericity and are not robust to small changes of the problem or to side-constraints, which requires the development of a brand new algorithm for each particular scheduling problem.

Nevertheless, this focus on pure problems motivated a lot of research on complexity results and polynomial-time approximation algorithms for different scheduling problems. This explains why this line of research has been and still is particularly prolific.

It is noteworthy that some of these algorithms have been made generic and are heavily used in more recent technologies, notably in propagators and as primal heuristics.

Another line of research in this community is the use of (mixed-integer) linear programming in conjunction with metaheuristics to solve scheduling problems. In spite of a number of major successes, this technology is often ill-suited since most resource constraints (such as the cumulative constraint) do not fit this model well. On the one hand, these constraints are difficult to linearize (they require at least a quadratic number of variables and have weak linear relaxations) and result in MIP formulations which do not scale. On the other hand, they are slow to evaluate and break the metaheuristics' performance. In spite of the genericity of mixed-integer linear programming and the fact that it is the best choice to tackle many different types of optimization problems, it is often not the technology of choice for scheduling problems.

For more information about the history of scheduling, we refer the reader to [Her06].

1.2.2 Constraint Programming

A different approach was initiated in the early 1990s with the first generation of constraint-programming libraries for scheduling, such as [AB93, Nui94] and Ilog Scheduler. Constraint programming finds its origins in the 1960s in AI and in Human Computer Interaction research, and was then used from the 1970s in combinatorial optimization, notably with Prolog. These libraries provided a generic language and propagators for each constraint, but the branching strategy was to be written by the final user, and had to be finely tuned to the model. Even though the distinction between model and search has always been clear in theory in CP, the practical necessity and difficulty of writing a search algorithm meant that the separation between the model and the ad-hoc search was sometimes blurred since information about the problem was embedded within the search algorithm.

Moreover, most of these languages and solvers were not tuned for scheduling problems, which was treated like any other combinatorial optimization problem.

It should be noted that, independently of scheduling, constraint programming is finding even more widespread applications today in different fields of computer science, for example with a recent application to cryptanalysis [GMS16].

1.2.3 Mathematical Optimization

A further step was taken in 2007 with the release of CP Optimizer 2.0, which incorporates many ideas from the Optimization community, and offers for the first time a *model & run* platform for scheduling problems. The principle of

model & run is to enable the user to solve the problem by modeling it in an appropriate language, without having to write algorithmic code, and by leaving the solving process to the optimization engine. This is for example what is done when using a MIP solver. We explain model & run further in the following subsection.

This was achieved for scheduling through the introduction of two new key technologies. First, a generic modeling language for scheduling problems was designed. This language is based on the notion of optional time intervals. These time intervals are used to represent activities of the schedule, as well as other temporal entities: union of activities, availability period of resources, alternative tasks, etc. This language can also natively represent several types of complex resources (temporal relations between tasks, cumulative resources, state resources, etc), which makes it expressive enough to model most common scheduling problems, yet concise.

Moreover, this language is algebraic: all quantities can be expressions of decision variables. Thanks to the "unreasonable efficiency of mathematics", this enables very powerful combinations of constraints, which enables to expose less than ten types of constraints to the user, as opposed to classical constraint programming systems, which have hundreds of types of constraints. This language will be explained in more detail in Subsection 2.4.1.

The second key technology is the introduction of a generic, automatic search [LG07], which had been a feature of mixed-integer linear programming solvers since the end of the 1980s. This automatic search relies on the principles of scheduling, since it is assumed that the problems which will be solved with CP Optimizer are dominated by their temporal structure. Moreover, the search engine exploits the semantics obtained from modeling in a high-level scheduling language, for example to distinguish between temporal and resource variables. The solving engine is based on many technologies in addition to constraint programming, such as local search, linear programming, scheduling heuristics, etc. Its functioning is explained in Section 2.5.

A lot of progress in the field of scheduling is due to the OR approach, but optimization and constraint programming made these solutions generic: constraint programming can indeed exploit OR algorithms while staying generic. This is why we can think of constraint programming not as a solving technology, but as an algorithmic integration platform, enabling different algorithms to collaborate, each of them working on partial problems.

As far as research in optimization is concerned, scheduling is today a major source of benchmarks to validate new computational techniques, with applications to Operations Research, Artificial Intelligence, AI planning, etc.

1.2.4 Model & Run

We must say a word here about the modeling principles when using such a Model & Run solver.

The progress of Operations Research has resulted in the development of tools which are intended to be simple enough to be used directly by the operational users, such as production engineers. This is due in a large part to the Model & Run approach in constraint programming, in contrary to previous approaches where part of the model was not written explicitely but actually concealed in the solving algorithm.

The combinatorial optimization paradigm is particularly fitting to that end. It is based upon an explicit model: clearly identified decision variables, constraints between these variables, and an objective function which must be optimized in accordance with the constraints.

This paradigm helps clarify the problem representation. This has several benefits for the user. By separating the modelling and solving processes, the user can focus on the model instead of writing a new search algorithm for each problem as is customary in the Operations Research approach. He thus benefits from the same flexibility as already offered by Mixed-Integer Programming solvers. As we will see below, tuning is done by modifying the model only, instead of having to make changes deep within the search algorithm.

To a certain extent, the model is even agnostic to the solving technology. This is the initial idea behind the Optimization Programming Language, which enables the same model to be solved either by a MIP engine or by a CP engine.

Nevertheless, using a high-level modeling language, as is possible with CP, has two main advantages. First, it allows to maintain some semantics on the problem, contrary to what can be expressed with a SAT or MIP model, for instance. In the case of constraint-based scheduling, this makes it possible to maintain a distinction between variables representing a temporal quantity and variables representing something else, such as a resource. This semantics can then be exploited by the solving engine through appropriate search techniques. Second, this high-level language also helps the user structure the model by providing her with a language which is designed specifically for scheduling problems instead of letting her write everything in the form of, for example, linear inequalities as is done when using a MIP engine. This allows for a much more compact and natural representation (see Subsection 2.4.1).

We now go into the detail of the different questions to keep in mind when writing an optimization model. As a reminder, a model is generally defined as being an instance of a theory (m) which satisfactorily reproduces the characteristics of the world (φ). Modeling is thus an art as much as a science, since it requires skillful tradeoffs between the expressiveness of the theory and fidelity to

the world characteristics. All models are simplifications and idealizations of the real phenomena they describe and, depending on the purpose, some truths on the world, certain aspects, and effects of different magnitudes are emphasized, approximated, or neglected. The 20^{th} century statistician George Box famously wrote on this subject: "All models are wrong, but some are useful". The Model & Run approach, with its explicit statement of the model, helps to control these gaps.

Having an explicit representation of the problem in the form of a model is of additional interest. Indeed, in contrary to the implicit assumption which we sometimes encounter in the community, that there exists a canonical model which perfectly describes the real-world problem, we believe that there are actually three steps to consider if one considers the question of solving an industrial optimization problem in its globality. After choosing an appropriate modeling language for the problem, we must model the problem in a modeling language, and pass it to an optimization engine. The engine will then find a solution to this model. The final step consists in implementing this solution in the real world, for instance in the full industrial context.

A common example of the subtlety of implementing the solution to a model to the real world appears in manpower scheduling, where it is easy to make the mistake of overprescribing the solution. In most manpower scheduling models one will naturally write indeed, each shift will be assigned to a particular worker. Actually, it is often unimportant to the management if two workers prefer to swap their shifts as long as they both have the required skills. In this case, any good implementation of the optimization results into the real world should be flexible enough to provide the possibility for employees to exchange their slots.

Thus, the model is not just an encoding of the problem data, but a simplified and modified version. This approach to solving an optimization problem is summarized on Figure 1.2.

In this framework, there are three sources of gaps between the world and the solution found by the solver to the model. They reside in the modeling limitations of the language, in the ability of the solving engine to work efficiently with the model provided, and in the gap between the best solution found by the solver and the mathematically optimal solution to the model (optimality gap). Let us detail each of them now.

The first source of gap comes from the lack of expressivity of the modeling language. For example, the available constraints may not be an exact match for the real constraints. Non-linearities might have to be linearized, if using a MIP solver. The need to keep the model compact can also be a limit to its precision.

The second source of gap, between the language and the solving engine,

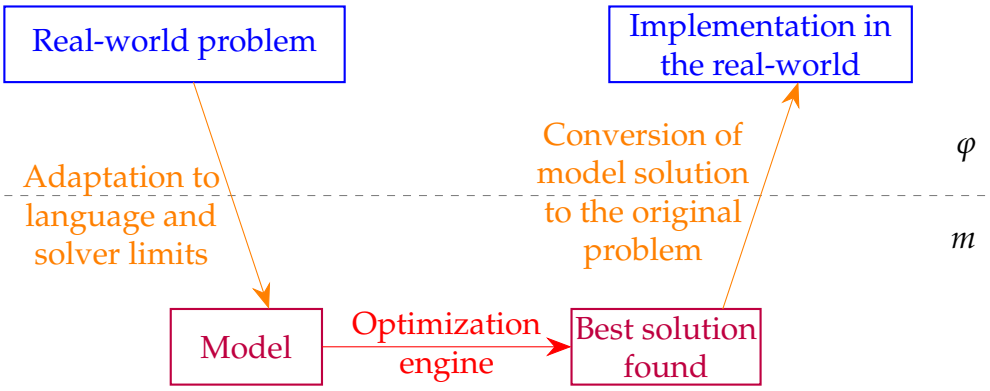


Figure 1.2: Steps to solving an optimization problem

comes from the need to balance an expressive language with the possibilities and needs of the solver. For instance, there may be different possible formulations with differing properties and efficiency. We might also have to explicitely model (or not, depending on the situation), redundant constraints or symmetry breaking. Of course, the higher the genericity of the modeling language and solver, the closer one can be to reality when modeling. So an engine design goal is to try as much as possible not to limit expressivity because of solving engine limitations, so that the user can deal only with the model instead of fine-tuning the solver to her particular problem.

Finally, the last source of gap is the optimality gap, that is the gap between the best solution found by the solving engine in a limited time, and the optimal solution to the problem. Since we are trying to solve NP-hard problems with limited resource, we will always encounter problems which we cannot solve to optimality in a given time (no free lunch theorem). In practice, it is possible to narrow this gap somewhat by giving the solver more time, on a more powerful computer, with more finely tuned engine parameters, and mostly with the continuous improvement of solving engines.

1.3 Contributions of this thesis

We focus in this thesis on offline, deterministic, non-preemptive scheduling problems, solved within a constraint programming framework. Offline means that the instance data is entirely known over the whole time horizon before we start making decisions, contrary to online problems where decisions have to be made without fully knowing what comes ahead. Deterministic means that the instance data is supposed to be perfectly accurate, in contrast to problems involving uncertain data, such as different possible scenarios or stochastic data

(see Subsection 2.4.2). Finally, non-preemptive means that the activities have to be processed without being interrupted once they have started. In other words, the end time of an activity is always equal to its start time plus its processing time.

In this context, our topic of interest is the cumulative constraint, in relation with temporal constraints. Our two main contributions are the Cumulative Strengthening and the Fast Energy Reasoning. The Cumulative Strengthening is a way of computing efficient reformulations of the cumulative constraint which strengthen many existing propagations and lower bounds. The Fast Energy Reasoning is a new algorithm for the Energy Reasoning propagation for the cumulative constraint, bringing its algorithmic complexity down from $O(n^3)$ to $O(n^2 \log n)$. These two contributions rely on duality results and geometric insights. They were both implemented on top of CP Optimizer and evaluated for their practical impact.

These contributions are related to the evolution of the field and of solvers technology as outlined in the previous section. Indeed, they reinforce the automatic search even in difficult cases, and the possibility of using the solver as a black box.

The reminder of this thesis is organized as follows. In Chapter 2, we review the basic principles of constraint-based scheduling. Since our work was implemented with CP Optimizer, we also present its basic design principles and the basics of automatic search. In Chapter 3, we introduce the cumulative constraint, study some of its properties, give examples of applications in industrial models, and present state of the art propagations and ways of computing lower bounds. Then we introduce the Cumulative Strengthening in Chapter 4. This chapter was already published in a shorter form as [BB14] and as a journal version in [BB17a]. In Chapter 5 we present the Fast Energy Reasoning algorithm. This chapter was already published in a much shorter version as [Bon16]. Finally, Chapter 6 is a conclusion on this work and an outlook on possible extensions.

Chapter 2

Scheduling with constraint programming

This chapter introduces the use of constraint programming to model and solve scheduling problems, which is the framework in which our work takes place. Section 2.1 presents scheduling techniques which found their origin in Operations Research. In Section 2.2, we briefly mention the different generic optimization tools which have been applied to scheduling. Section 2.3 is a brief review of constraint programming, and Section 2.4 focuses on its application to scheduling. Finally, the design principles of CP Optimizer and of the automatic search are exposed in Section 2.5.

2.1 Machine scheduling

Most of the early research on scheduling under resource constraints, in the Operations Research community, was done in the context of machine scheduling, where resources correspond to machines. This field is prolific and a large part of the research on scheduling is still performed in this context.

We present the most common problems in this field in Subsection 2.1.1, briefly reference general extensions of these problems and show how they are classified to form a theory of machine scheduling in Subsection 2.1.2, and present a selection of algorithms for this field in Subsection 2.1.3.

These problems are all special cases of the RCPSP, defined below in Section 3.2, and the constraint-based scheduling approach subsumes these techniques, but they form the algorithmic origin of a part of a constraint-based scheduling engine.

2.1.1 Machine and shop scheduling problems

In machine scheduling problems, we only deal with specific resources, namely machines, which correspond to actual machines in a workshop or factory. The problem consists of n jobs $J_1, \ldots, J_n$ which must be scheduled on the machines. These jobs are non-preemptive, meaning that their processing cannot be paused or interrupted once started, the problem is deterministic, meaning that we assume that there is no uncertainty on the problem values, and offline, which means that all information is available before solving the problem.

In the simplest cases, the *single machine problem*s, we have only one machine. This machine can only process one job at a time, we must therefore find a total order on the jobs.

In *parallel machine problems*, we now have m machines $M_1, \ldots, M_m$ available, which can each process one job at a time. The machines are said to be *identical* if the jobs can be processed on every machine and the processing time is the same on each machine. They are said to be *unrelated* if the processing time depends on the machine on which a job is processed. Certain jobs can also be performed only on certain machines.

In their most general form, *shop scheduling problems* consist of m machines $M_1, \ldots, M_m$, and each job J_i consists of $n(i)$ operations $O_{i,1}, \ldots, O_{i,n(i)}$. Each operation $O_{i,j}$ must be processed on a specific machine $\mu_{i,j}$, two operations of the same job can not be processed simultaneously, and a machine can process a single operation at a time. In this most general form, this problem is called *open-shop*. We mention two important special cases: the *job-shop*, where the operations of a job are constrained by a chain of precedence constraints, giving a total order on the processing dates of the operations of a job and the *flow-shop*, which is a special job-shop where each job consists of the same operations, which must be performed in the same order for each job.

For more information about these models, we refer the reader to Section 1.2 of [BK12] and Part I of [Pin12].

2.1.2 Complexity classes

The basic problems we defined in the previous subsection accept many variants in terms of additional constraints on the tasks (release dates, due dates and precedences), machines capabilities, objective functions (makespan, lateness, throughput, machine cost, etc.), and an algorithm developed for one scheduling problem P can often be applied to another problem Q, in particular to all special cases of P. When this is the case, we say that Q *reduces* to P. This provides a hierarchy of problems, solving techniques and complexity results. Formalizing

this hierarchy provides the beginning of a theory of scheduling complexity. The most common classification in this context is that of Graham, also called $\alpha \mid \beta \mid \gamma$.

α corresponds to the machine environment, that is the resources: single machine, several machines which can be identical, uniform, or unrelated, or shop problem: flow shop F, job shop J or open shop O.

β represents the job characteristics: precedences between the tasks, release dates r_i, deadlines d_i and durations p_i.

γ is the optimality criterion or objective function. Here, C_i denotes the completion time of job i, and T_i is its tardiness: given a desirable due date δ_i for each job i, $T_i = \max(0, C_i - \delta_i)$ is the lateness of this job with respect to the due date. Among the most common optimality criteria are the makespan $\max_i C_i$, the total weighted flow time or sum of weighted completion times $\sum_i w_i C_i$, the maximum tardiness $\max_i T_i$ and the total weighted tardiness $\sum_i w_i T_i$.

This classification is restricted to the class of scheduling problems mentioned above (machine and shop scheduling), and most problems considered in this thesis do not belong to the $\alpha \mid \beta \mid \gamma$ classification. Nevertheless, the study of this field resulted in finding specific algorithms, as well as approximation schemes.

We refer the reader to the elegant and exhaustive Scheduling Zoo website [**?**] for more information on this subfield.

2.1.3 Basic algorithms

We present here a small selection of classical results, which have been inspirational for further developments in cumulative scheduling. We refer the reader to [Pin12] for more information on this subfield of scheduling.

- The first of these results concerns minimizing the sum of completion times of tasks scheduled on one machine, or $1 \mid\mid \sum_i C_i$. This objective is also called the average flow time. In this case, the Shortest Processing Time first (SPT) rule results in an optimal solution: tasks should be run in the order of increasing durations p_i. A weighted version of this result exists: if each task has a weight w_i and the objective is to minimize $\sum_i w_i C_i$, then the tasks should be run in the order of increasing $\frac{p_i}{w_i}$ values.

- In the case of $1 \mid\mid L_{\max}$, that is when the tasks are again processed on one machine, have a due date d_i, and the objective is to minimize the maximum lateness $\max_i (C_i - d_i)$ where the C_i are the actual completion dates, the optimal solution is obtained by using the Earliest Due Date (EDD)

rule, also called Jackson's rule. The tasks should be run in the order of increasing due dates.

- Finally, we present the $Pm \mid prmp \mid C_{\max}$ problem, that is minimizing the makespan of a set of jobs which can be executed on m identical parallel machines now, and can be preempted (meaning that their execution can be interrupted before it is complete and restarted later, possibly on a different machine). In this case, McNaughton's algorithm (Theorem 3.1 of [McN59]) gives an optimal schedule.
  Notice that $C^*_{\max} = \max\left(\max_{j=1}^{n} p_j, \sum_{j=1}^{n} \frac{p_j}{m}\right)$ gives a lower bound to this schedule. We then schedule the jobs arbitrarily on a single machine. The makespan of this schedule is $\sum_{j=1}^{n} p_j \leq mC^*_{\max}$.
  Finally, we split this single machine schedule into m equal parts of length $C^*_{\max}$ and schedule it on the parallel machines: the interval $[0, C^*_{\max}]$ goes on machine 1, $[C^*_{\max}, 2C^*_{\max}]$ goes on machine 2, ..., $[(m-1)C^*_{\max}, mC^*_{\max}]$ goes on machine m.
  This schedule is feasible. Indeed, part of a task may appear at the end of the schedule for machine i and at the beginning of the schedule for machine $i+1$. Since no task is longer than $C^*_{\max}$ and preemptions are allowed, this gives a valid schedule. And since this schedule has length $C^*_{\max}$ which is a lower bound, it is optimal.

2.2 Combinatorial optimization

A way to generalize the approaches seen above and find more generic algorithms to solve scheduling problems is to express scheduling problems in a combinatorial optimization framework.

In their most general form, combinatorial optimization algorithms optimize a function $c : \mathcal{F} \to \mathbb{R}$, defined on a finite set $\mathcal{F}$ of feasible solutions. The different techniques we will present apply to different special cases of this general definition, depending on the structure of $\mathcal{F}$.

As we will see in the following section, Constraint Programming gives a framework to call on these different techniques (and others!) to solve subproblems of the main problem and integrate the results thus obtained.

The following techniques have all been used successfully within a CP framework to solve scheduling problems.

2.2.1 Linear Programming

Linear programs are expressed as the problem of optimizing (maximizing or minimizing) a linear function defined over a set of continuous (floating-point, typically) variables, which must satisfy a set of linear inequalities over these variables:

$$\begin{aligned} \max \quad & \sum_{j=1}^{n} c_j \mathbf{x_j} \\ \text{s.t.} \quad \forall i \in [1,m] \quad & \sum_{j=1}^{n} a_{ij} \mathbf{x_j} \leq b_i \\ \forall j \in [1,n] \quad & \mathbf{x_j} \geq 0 \end{aligned}$$

There exist different families of algorithms to solve linear programs, the most common among them being simplex algorithms and interior points methods. Even though no one has yet designed a simplex algorithm of subexponential complexity, interior points methods have a polynomial time complexity so the problem of linear programming itself is polynomial. Without going into the details here, we must say that there exists a very rich geometrical theory about the structures linear programming operates on. An active research topic in linear programming consists in the design of simplex algorithms with a provably polynomial complexity.

Moreover, the paradigm of linear programming is very broadly used in optimization and operations research, due to its versatility. Indeed, many industrial problems have a natural representation under the form of a linear program, notably problems of assignment of resources to productions: an interpretation of a linear program consists in seeing the variables as production quantities of different items, while the constraints represent limits on the available resources.

This versatility has generated an intense industrial interest since the end of the 1940s into the applications and the effective computer solving of linear programs. Very efficient codes are available today, for example CPLEX which is developed by IBM Ilog along with CP Optimizer.

A technique which is very commonly used in conjunction with linear programming, notably when we have to represent combinations, patterns, or possible subsets of a certain set, is delayed column generation. Indeed, one of the drawbacks of linear programming is that many problems of interest have a superpolynomial (typically exponential) number of variables in their representation as a linear program, even though only a small number of these variables (no more than the number of constraints when the program is written in standard form, according to duality theory) will have a nonzero value. The principle

of delayed column generation to solve these very large problems is to consider only a subset of the variables, to solve this partial problem (called the master) and then to check with an adhoc algorithm, depending on the problem in question (called the slave), if setting one of the missing variables to a nonzero value could result in an improved solution. If this is the case, we add this variable (column) to the master problem and repeat the process.

In practice, only a small number of variables will be added to the master problem. In spite of the additional complexity of solving many different versions of the master problem and having to solve the slave problems, the full procedure can typically be several orders of magnitude faster than solving the main problem directly (if it fits into memory at all)!

2.2.2 Mixed Integer Programming

Mixed Integer Linear Programming (typically called MIP) is similar to linear programming, with the difference that a subset J of the variables can only take discrete values (in general Boolean values).

$$\begin{aligned} \max \quad & & \sum_{j=1}^{n} c_j \mathbf{x_j} \\ \text{s.t.} \quad & \forall i \in [1,m] & \sum_{j=1}^{n} a_{ij} \mathbf{x_j} \leq b_i \\ & \forall j \in [1,n] & \mathbf{x_j} \geq 0 \\ & \forall j \in J & \mathbf{x_j} \in \mathbb{N} \end{aligned}$$

The main idea behind solving mixed integer linear programs is to use a branch-and-bound technique by recursively splitting the domain into smaller subdomains and solving the continuous relaxation of the problem, i.e. the problem without the constraint that some of the variables can only take integer values (which is thus a linear program). If the optimal solution x^* to the linear relaxation is such that all components in J have integer values, then we have an optimal mixed integer solution. Otherwise, a component x_i^* with $i \in J$ of x^* is fractional and two subproblems can be generated, respectively with the additional constraints $x_i \geq \lfloor x_i^* \rfloor + 1$ and $x_i \leq \lfloor x_i^* \rfloor$, and we can recurse. The optimal value of the continuous relaxation gives a primal bound on the optimal value of a mixed integer solution.

In addition, so-called cuts can be generated automatically. They are additional linear inequalities which are generated in such a way that the integer solutions are still feasible, but not necessarily the fractional solutions. The cuts

reduce the size of the search space and increase the likelihood that the continuous relaxation will find an integer solution.

There has also been a tremendous amount of engineering into making MIP solvers generic and efficient. Moreover, with the addition of integer variables to the linear programming model, all types of combinatorial problems can be modeled in the MIP framework which makes it extremely versatile. Among its drawbacks are the facts that no semantics is kept on the model, and the fact that almost all scheduling problems have MIP representations whose size is exponential in the problem size, making this technology inefficient when compared to constraint programming. We discuss MIP representations of scheduling problems later in Section 3.5.

2.2.3 SAT

A Boolean formula is defined over Boolean variables, which can only take the values True or False. A literal l is a variable a or its negation $\neg a$ (the negation of a is True whenever a is False and vice versa). A clause $c_i = (l_{i,1} \vee \ldots \vee l_{i,n_i})$ is a disjunction of literals (True if any of the literals is True). Finally, a CNF (conjonctive normal form) formula is a conjunction $c_1 \wedge \ldots \wedge c_m$ of clauses (True if all the literals are True). Of course, the same variable can appear in several literals in one formula.

The SAT problem consists in determining whether there exists an assignment of values to the variables of a CNF formula such that that formula is True. This problem is the prototypical NP-complete problem (Cook's theorem, 1971) and it is very generic: many combinatorial optimization problems can be naturally modeled as a SAT problem.

For this reason, very efficient algorithms have been developped: first with DPLL-based solvers which perform a recursive search over the possible literal values. These solvers rely heavily on tailored data structures which enable very efficient unit clause propagation (if all literals but one are False in one clause, then set this literal to True everywhere in the formula).

A new generation of solvers appeared in the last decade, called clause learning solvers, which make use of the large quantity of RAM available in modern machines to generate a new clause to the formula each time they encounter a failure, so as not to explore this part of the search space again.

A lot of engineering has been done in this field, motivated by SAT solver competitions, and very efficient programs are available. Fruitful ideas have also been exchanged with the CP community.

2.2.4 Dynamic Programming

This technique can be used for problems which satisfy the optimal substructure property (sometimes also called Bellman's optimality principle), which states that an optimal solution can be recursively constructed from optimal solutions to subproblems of the same type.

When this property applies, we can memorize the solutions to the subproblems in order not to recompute them every time they appear in the resolution of a more general subproblem.

A common example in scheduling is the Weighted Interval Scheduling Problem: given a set J of optional jobs j, each with a start time s_j, an end time e_j and a weight w_j, the goal is to compute the maximum sum $f(J)$ of the weights of non-overlapping jobs. In other words, we have only one machine and we must choose a set of compatible tasks which will yield the highest possible profit.

We can see that if we assume that job i is executed in an optimal solution, then we can solve the problem separately over jobs which end before s_i and over jobs which start after e_i, and sum the results to obtain the overall optimal solution, so for any $K \subset J$,

$$f(K) = \max_{i \in K} w_i + f(\{j \in K : e_j \leq s_i\}) + f(\{j \in K : s_j \geq e_i\})$$

and the computation of $f(K)$ is reduced to the computation of f for non-overlapping subsets of K, making this problem suitable for the dynamic programming framework.

2.2.5 Heuristics

When complete methods are ineffective on a class of problems, we have to rely on heuristics algorithms. These algorithms make different approximations on the structure of the set of solutions in the hope of finding good enough feasible solutions to very difficult problems, but the solutions obtained and are not guaranteed to be optimal.

We generally use the word heuristics for algorithms which are tailored to a very particular problem and metaheuristics for techniques which are more general.

There are two general ways of using (meta)heuristics in combinatorial optimization and in scheduling in particular. The first one is local search in which one starts from a feasible solution, possibly of poor quality and which can be obtained via various means, and gradually transform it into a good one via a series of small, incremental steps, from feasible solution to slightly better feasible

solution. This approach works well if the space of feasible solutions is densily connected for the heuristic algorithm in use.

The second way of using heuristics is to make use of an indirect representation of the problem (which can be a simplified version of the problem, or a very different problem, but which can be effectively transformed into a full solution) and to use the metaheuristics on the indirect representation. If the indirect representation effectively captures the difficult part of the problem (sometimes called problem kernel) and the transformation into a full solution is easy, this can be a very powerful technique. See also Subsections 2.5.1 and 2.5.3.3.

Among the most common heuristics and metaheuristics one can find are genetic algorithms, ant colony optimization, hill climbing, simulated annealing, and tabu search. All of them have been used successfully for scheduling.

2.3 Constraint programming

Constraint programming is based on the declaration of a combinatorial problem in the form of decision variables, linked together by constraints, and an objective function of these variables, to be optimized.

The basic technique for solving these problems is a tree search of the search space by separation-evaluation, associated at each node of the tree with a stage of constraint propagation. Propagation consists in checking, with the aid of specific algorithms, necessary conditions on the compatibility of the potential values that the variables can take (domains), in order to eliminate some of them and thus reduce the size of the space of search to explore downstream of the current node.

As such, constraint programming can be seen as a language, since its directory of decision variables and constraints offers a modeling language, but also, on the solving side, as an algorithms integration framework allowing the problem to be broken down into a set of very orthogonal polynomial subproblems, through search heuristics and propagation algorithms, each working on a well-structured subset of the problem but pooling their results through variables domains (see below) and indirect representations (see Subsection 2.5.1). For example, a typical scheduling problem can be decomposed into a network of precedence constraints, on which we can reason in polynomial time, and into resource constraints on which we can reason using polynomial complexity propagation algorithms (see Section 3.4).

Formally, a *constraint programming problem* consists of a set of *decision variables*, a set of *constraints* over these variables, and an *objective function*. Let us define these terms now.

The decision variables are the variables which must be determined by the solving procedure. They correspond to the operational decisions that have to be taken. Three types of decision variables are commonly encountered in constraint programming: integer, floating-point and time intervals (in scheduling engines). Two important cases of integer variables are variables representing boolean decisions or the choice of an element in a finite set. In constraint programming, decision variables take their value in a domain, which is a set of potential values. The domains will be reduced as the search goes along and potential values can be eliminated.

Constraints are compatibility conditions between the variables. They are typically either binary (they involve two variables) or global (they involve multiple variables, which gives more semantic information on the problem and enables remove inconsistencies more efficiently than by just using binary constraints). In practice, constraints are enforced by propagation algorithms, which remove incompatible values from the domains of variables.

Propagations are the keystone of constraint programming. A constraint is said to have a support if there is an assignment to each variable of a value of its domain, such that the constraint is satisfied. If a constraint has no support, the problem is infeasible. We call this a failure. If a value does no belong to a support for any constraint, we say that it is inconsistent and it can be removed from the variable domain. Propagation therefore consists in eliminating values from variable domains that are inconsistent with a constraint. For example, if task B must start after task A ends and task A can not finish before time $t = 7$, then we know that task B must start at or after time $t = 7$ and can eliminate smaller values from the domain of its earliest start time.

We use different propagation algorithms, which work on different constraints or groups of constraints and check different necessary conditions. Usually we execute them repeatedly until none of them is capable of reducing the domain anymore (we say we have reached the fixed point of propagation), or one of the domains becomes empty (in which case the problem has no solution). The result of propagation is therefore an equivalent problem, in the sense that it has the same solutions, but more consistent (smaller search space).

We distinguish several types of consistency. For example, we say that we have achieved arc consistency if all unsupported values have been removed from all domains. Arc consistency can be very costly to maintain, so we often resort to a weaker form of consistency such as bound consistency. In bound consistency, the domains are intervals, and we only make sure that the minimum and maximum of this domain is not unsupported. However in scheduling the constraints are so complex that even the bound consistency is often too expensive to maintain. We therefore only verify particular cases of BC, differ-

ent propagation algorithms for the same constraints having different ratios of computing time versus propagation power and being used in different contexts.

In practice, it is essential that propagations be incremental, that is, not all combinations of domains be re-studied from one node to the next, since branching decisions have a local impact. Incremental propagations only re-examine values that could have been impacted by the last branching decision.

As for the search algorithm, the tree search starts at each node by calling the propagations. Depending on whether or not a conflict is detected during this step, three outcomes are possible:

- If no conflict is detected and each variable has only one value in its domain, we found a feasible solution to the problem. We can compute the objective value for this solution and update the best solution found.
- If no conflict is detected but there are variables whose domain still has several values, we call a branching heuristic to divide the search space in two, and call the search recursively on the left node and then on the right node.
- If the propagation has detected a conflict (no value left in the domain of a variable), we backtrack, that is we return to the previous node, cancelling the last branching decision. If we are already at the root node and cannot backtrack, we have proven that the problem has no more solutions.

2.4 Constraint-based scheduling

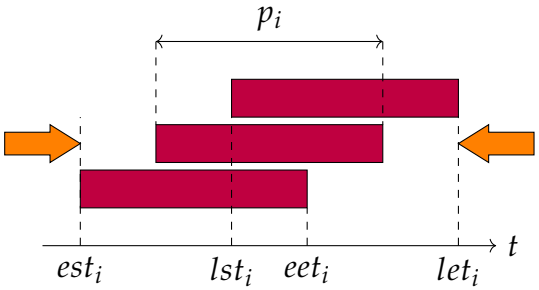


Figure 2.1: Bounds of a time interval in constraint-based scheduling

Constraint-based scheduling consists in solving in solving scheduling problems using a large part of the conceptual framework of constraint programming, enriched with scheduling-specific notions.

The contraint program thus possesses variables for the execution dates (start and end dates) of each task as well as for the resources use over time. It is crucial to use a scheduling-specific modeling language, instead of simply enriching an integer variables based constraint programming system with scheduling constraints. Indeed, the temporal structure, that is the distinction between the temporal and resource variables and constraints, is essential to the efficiency of the solving algorithms.

It is also essential, both for the solver and for the person writing the model, to have a small, compact model. To this end, the modeling language is algebraic, which means that all appearing quantities are themselves decision variables. This enables the engine to understand part of the problem semantics and to exploit scheduling concepts beyond what could be done in pure constraint programming.

2.4.1 Language overview

Let us now give a very brief overview of the CP Optimizer constraint language. More information can be found in [LR08, LRSV09].

The language is based on the notion of interval variables. These variables possess a presence status, whose domain contains the two values present and absent, as well as a start date and an end date. The domain of the start date ranges from the earliest start time to the latest start time, and the domain of the end date ranges from the earliest end time to the latest end time (see Figure 2.1). Decisions, made by propagation or during the search, will tend to reduce these ranges. The presence status is essential to facilitate modeling, for example when we have activities whose execution is optional, to model tasks which can be executed on alternative resources or in different modes, or through different processes.

Integer variables are also available, if needed, but the scheduling language is expressive enough that they are generally not needed.

Several families of constraints are available. Temporal constraints first, such as bounds on the start or end dates of intervals or more complex arithmetic expressions of those. Additionally, precedence constraints state that an interval can start only after another one ended, and optional delays between these two events can be specified.

Alternative modes can be modeled using the Alternative constraint, where a master interval is synchronized (same presence status, start and end dates) as one of several slave intervals representing the different modes, the presence status of the other slave intervals being set to absent.

A very useful constraint to model hierarchical processes is the Span, which states that a master interval should extend over all of time range covered by

slave intervals.

The disjunctive is a resource constraint: it models a unique resource used by several intervals, and forces them not to overlap. Since this yields a total order on the execution of these intervals, the sequence that we obtain can also be thought of as a temporal constraint, and constraints are available to manipulate it, such as Prev and Next (constraints on the intervals which should precede or follow another one).

The two main other resource constraints are the cumulative, which we will see in great detail in Chapter 3 and is the focus of this thesis, and the state constraint, which is used to model situations where a machine can perform different operations when it is in different states, or industrial ovens where certain tasks can only be performed within a certain temperature range.

An objective function can also be specified as a general function of the decision variables. This flexible design allows for very general objective functions, which can be a variant of the makespan or incorporate additional costs such as setups, lateness, as well as non-temporal costs such as resource allocation, non-execution, etc. Sometimes, the real objective is very difficult to express as a function to optimize, since the industrial objective is not to over-optimize every detail of the process at the expense of flexibility, but rather to smooth things out and minimize fits and starts. Modeling these objectives requires a great deal of expertise.

2.4.2 Assumption of determinism

Of course, not all scheduling problems conceivable can be modeled in this framework even if, in practice, a great breadth of problems fit [Lab09].

One important limitation to pay attention to is the assumption of determinism, which requires particular care. As first noticed by Fulkerson in [Ful62], in the case of uncertainties about the duration of certain tasks in a scheduling problem, using the expectation of the duration as a deterministic value will most of the time lead to a systematic underestimation of the total processing time for the whole project. This is in particular the case if the function f which gives the optimal value of the objective is a convex function of the processing times $\boldsymbol{p}$. Indeed by Jensen's inequality, $f(E(p_1), \ldots, E(p_n)) \leq E(f(p_1, ..., p_n))$. Notably for us, the RCPSP (see later Section 3.2) falls into that category if the objective function is regular (monotonous with the start and end times of the tasks), since a solution which is feasible with a certain vector of processing times $\boldsymbol{p}$ is obviously still feasible if the processing time of a task is reduced.

In practice, we resort in these cases to writing a robust model: different scenarios on the stochastic variables are sampled from the joint probability distribution and evaluated simultaneously, in the sense that the scheduling decisions

made must be compatible with all scenarios. The objective function also takes the different scenarios into account, for example optimizing for the worst-case. An interesting feature of constraint programming in the context of robust optimization is that we can use global constraints to join decisions taken for the different scenarios at a high-level. An example of such a constraint is the Isomorphism in CP Optimizer, which constrains two set of tasks to be executed in the same order.

2.5 Automatic search in CP Optimizer

CP Optimizer has a powerful automatic search algorithm. It presents several properties: it is generic (the same algorithm is used to solve all problems and does not require to be rewritten when the problem changes), complete (it is guaranteed that the optimal solution will be found or in other terms that all of the search space will eventually be traversed, even if this can take a very long time), deterministic (in spite of the use of randomness during the search and a parallel implementation, it is guaranteed that running the search several times on the same problem with the same parameters will yield the same solution), adaptative (its parameters are automatically adjusted during the solve to the particular instance characteristics) and robust (it is invariant with respect to a number of changes of the instance, for example a dilation of the time scale).

This automatic search relies on several techniques on top of Constraint Programming. Among them are Linear Programming, local methods, scheduling heuristics, machine learning, etc.

Reaching the objectives stated above requires several new ideas.

First, as we have shown above in Section 2.4, we use a scheduling-specific language, notably through the concept of optional interval variables. We also make a strong use of the fact that we work specifically on scheduling problems to guide the search through chronological branching techniques.

Second, indirect representations of the problem are used in many parts of the solving algorithm.

We will detail these principles in the following subsections.

2.5.1 Global variables and indirect representations

When working on a difficult problem, it is often advantageous to change its representation, for example as an instance of a problem we can already solve. The main principle behind finding efficient representations is to focus on the parts of the problem that form the core of its difficulty.

CP Optimizer makes a heavy use of global and indirect representations of the problem. Some are explicitely given in the model, other are computed or inferred from binary constraints or from deductions made during the search.

These indirect representations are used both by the propagations and by the search algorithms, and actually everywhere a decision is taken. They help make the algorithm robust by considering more global structures when taking decisions than just the constraints independently from each other. These representations play a role similar to that of duality and cut generation in MIP engines.

Some of these representations are analogous to global constraints, such as the temporal and logical networks, and could even have been expressed as such in the language instead of being aggregated from binary constraints. This choice was made out of convenience for the user.

Propagation is performed both on the model variables and on these representations.

A few examples of these indirect representations follow.

The temporal network is a fundamental concept in the algorithmics of scheduling and comes from AI planning. The principle is to aggregate all precedence constraints in a single graph, which exhibits much more of the temporal structure of the problem than just considering the pairwise relations between intervals. This graph is dynamically updated using pairwise relations discovered during the search.

The logical network plays a similar role and aggregates all the constraints between presence statuses of interval variables. It is beneficial to consider these relations globally since the problem of propagating all these statuses globally is equivalent to 2-SAT, for which efficient algorithms exist. These two networks are described in more details in [LR08].

Many propagators on global constraints, such as the timetable, the edge-finder, etc, maintain incremental data structures along the search tree, which constitue global and relaxed representations of a part of the problem.

Finally, some local search algorithms, such as the Genetic Algorithm, or list scheduling, rely on an indirect representation in the form of a simplified problem which can then be expanded into the full problem. Please see Subsection 2.5.3.3 for more information.

Other examples of indirect representations will be shown in the rest of this chapter, such as the linear relaxation, POS in Large Neighborhood Search and partial enumeration of interval variables in FDS.

2.5.2 Branching strategies

In a tree search, branching strategies are the heuristics which choose the variable whose domain will be split, as well as the partition of that domain that will

be explored in the left and right branches. They are also called, in scheduling and depending on the context, dates fixing algorithms or completion strategies.

They must be worked out in a strategic fashion, since a large fraction of the overall engine performance depends on their ability to drive the search quickly (that is, using mostly left branches, which are explored first) towards a possible solution. Thus, in scheduling, we use chronological branching strategies, which make use of the temporal nature of the problem. The main idea of these strategies is to select as branching variable that whose value will be chosen as small as possible (or as large as possible in the case of a reverse chronology). This combines well with the propagations since it densifies the space occupied locally, at the beginning of the schedule.

2.5.2.1 Set Times

The objective function f is said to be *regular* if for any two schedules S and S' such that all dates in S are smaller or equal to the dates in S', $f(S) \leq f(S')$. It intuitively means that all other things being equal, we should schedule activities as early as possible. Many common objectives, such as makespan, weighted flow time, maximum lateness, total weighted tardiness are regular but not earliness-tardiness, for example.

When the problem has a regular objective, we can use the Set Times branching rule, which is a form of chronological backtracking and makes use of the intuition that tasks should try to be scheduled as early as possible. To the extent of our knowledge, the Set Times rule was first published in [PCVG94]. Other explanations can be found in [GLN05, LR14], but many other variants exist.

The basic principle of Set Times is to try in the left branch to set the activity with smallest earliest starting time to start at its *est*. If a failure occurs, we say that the activity is *postponed*, and we know that we cannot start this activity at its *est*. We will not select this task anymore until its *est* has been changed further down in the search tree.

2.5.2.2 Temporal Linear Relaxation

When the objective is irregular, such as with earliness costs, variable activities duration or delays between activities, or when it includes non-temporal costs such as setup costs or non-execution costs, scheduling can be extremely difficult, even in the absence of resource constraints. The chronological branching heuristics such as Set Times are poorly suited and do not perform well in this setting.

In this case, CP Optimizer uses the solution to a linear programming relaxation of this instance as an oracle, through a branching heuristics, as explained

in more details in [LR14]. The idea is to combine the ability of linear programming to solve complex allocation problems, as long as they are linear, with the ability of CP to solve minute details of activities placement and resource consumption. Resources do not need to have a very tight linear relaxation, and since the Temporal Linear Relaxation is used in the context of Large Neighborhood Search, we will make use of the POS (see below in Subsection 2.5.3.2) as a strong and already available linearization of the resource constraints.

The Temporal Linear Relaxation is based on a model whose decisions variables are continous: $x_i \in [0,1]$ represents the presence value of interval i, s_i its start date and e_i its end date. Several constraints are already linear, such as logical constraints (logical relations between the presence statuses of several intervals) and precedence constraints. Moreover, since the TLR is tightly integrated within the Large Neighborhood Search, most of the complex resource constraints have already been linearized into precedence constraints in the Partial Order Schedule. Arithmetical expressions are relaxed into convex piecewise linear expressions. Some additional inequalities are added to the linear formulation for additional strength, such as for alternative constraints, or the cumulative constraint for which a global energy bound is computed. We refer the reader to the paper on TLR for more details on these.

Finally, the branching heuristics based on the Temporal Linear Relaxation works in a similar fashion to the Set Times heuristics seen above, except that it uses the values computed for s_i and e_i as references instead of est_i. For the presence value of interval i, the probability of setting it present in the left branch is equal to x_i.

The TLR is also used to compute a global lower bound on the problem, and the reduced costs are used to reduce the variable domains when possible.

Using these principles, the Temporal Linear Relaxation gives another example of the possibility to integrate algorithms of a different nature thanks to Constraint Programming.

2.5.3 Search strategies

As we will see below, for example with the RCPSP in Subsection 3.2.4, scheduling problems are NP-hard in general, so the goal of a commercial solver is to look for as good as possible solutions in a limited computational time.

To achieve this goal, the search follows these general principles:

- We should avoid spending time exploring parts of the search space where no solution lies. In constraint programming, this is done through constraint propagation (see below in Subsection 2.5.4) and the use of dominance rules when branching (as seen above in Subsection 2.5.2).

- When a good solution is found, the search is intensified around it, since this solution probably has properties that are desirable for other good solutions.
- There should be no large regions of the search space left unexplored, different techniques are used to diversify the search.
- The horizon can be very large in scheduling, so enumerating the temporal variables should be avoided. Reasonings should be performed on events instead.
- All searches are regularly restarted. Restart is a very fundamental concept in search space exploration. The goal is to avoid getting trapped in local optima by regularly restarting the search from scratch, hoping to explore a different region of the search space. Some information can be kept from previous searches, such as no-goods and learnt parameters on heuristics and properties of the search tree. More information on restarts can be found in [LSZ93, KHR+02].
- The search is self-adapting. As many internal parameters as possible are learnt dynamically during the search.
- Since CP Optimizer is intended to be used in an industrial setting with limited time available, the emphasis is set on finding good solutions (improving the primal bound). Time is spent improving the dual bound only when it is assumed that we already know the optimal solution or an exhaustive search space exploration is needed.

In line with these principles, here are the general steps performed by CP Optimizer's generic search algorithm:

0. A presolve step is run. Different transformations are performed on the model to simplify it, remove some modeling inefficiencies, and make it more suitable for the next steps.
1. A first feasible solution is searched, using a branch & bound tree with aggressive heuristics which emphasize finding a feasible solution over optimizing the solution quality.
2. An initial dual bound is computed by two means. First by running a binary search on its value with a tree search of limited size, second by reusing the value found by the linear relaxation mentioned above. This dual bound will be used in the branch & bound search and to measure the quality of the solutions found.

3. The main search is started. Different strategies are available, and are generally run in parallel, with a different CPU time ratio so as to favor the most promising ones. We detail the available search strategies in the rest of this subsection.

2.5.3.1 Depth-First Search

The Depth-First Search method is the most basic search algorithm in CP Optimizer. It consists in a simple branch & bound exploration of the search tree. The branching heuristics is the Set Times if the objective is regular, or the linear relaxation otherwise.

This algorithm is complete (all feasible solutions are explored), but the cost of doing so can be tremendous. In practice, it is mostly used for debugging and for counting the number of feasible solutions to a problem.

2.5.3.2 Large Neighborhood Search

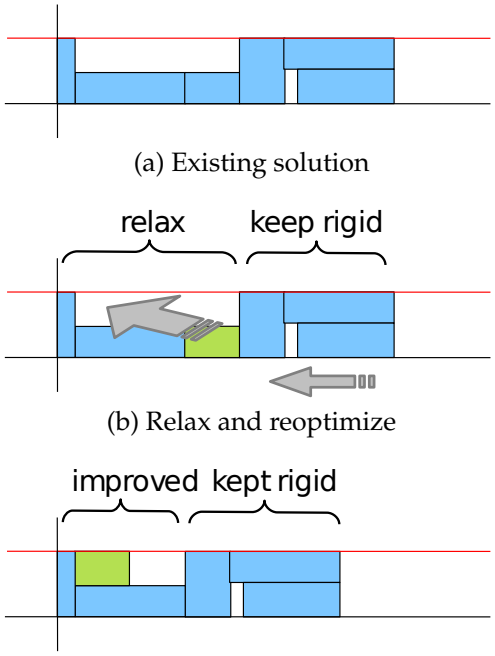
The Large Neighborhood Search is the main search strategy used by CP Optimizer, once at least one feasible solution to the problem has been found. This strategy was introduced in [Sha98] and was adapted to scheduling problems and expanded in [Lab05, LG07]. It can be thought of as a local search based metaheuristics on top of the tree search. The principle is to relax a part of the incumbent solution, to reoptimize, and to repeat the process, as shown on Figure 2.2. We will hereunder describe the main ideas of LNS.

Since LNS is a local search algorithm, we need to find a feasible solution first. This is done using a tree search, with parameters which are optimized to quickly find a feasible solution, at the expense maybe of its quality.

Once that first solution is available, LNS itself can start and will loop over relaxing a part of the current solution, and completing it into a full solution. Relaxing a part of the solution is done by keeping some structure on the rest of the solution and reoptimizing the whole problem with this additional structure. The structure that is kept is called a Partial Order Schedule and was introduced in [PSCO04]. It consists in a graph whose edges are precedence constraints between activities, with the property that all schedules which are feasible with respect to the POS also satisfy most resource constraints of the problem (except for very particular situations such as a minimum level constraint in a cumul, for which the POS might be underconstrained).

Of course, the POS is over-constrained with respect to the original problem since it encodes part of the incumbent solution.

Computation of the POS is done resource by resource. For example, for a sequence, the POS contains all precedences from one activity in the sequence



(a) Existing solution

(b) Relax and reoptimize

(c) Improved solution

Figure 2.2: Schedule Improvement with Large Neighborhood Search (Figure courtesy of Philippe Laborie)

to the next. For a state resource, the POS contains all precedences between activities requiring incompatible states, such that one of them is executed fully before the other. For a cumulative resource, the computation is slightly more involved and is described in details in [LG07].

The part of the solution which will be relaxed can be chosen by different heuristics, each depending on parameters. For instance, one such heuristics will randomly select activities to be relaxed with a certain probability. Another one will relax all activities which start in a certain time window. The position and size of that time window are parameters of that heuristics.

Once we have chosen a fragment to relax, all edges in the POS involving activities from the fragment are removed, and this new problem is solved. The additional constraints provided by the POS will result in a much smaller search space.

The LNS is self-adaptating, as are most algorithms in CP Optimizer, in the sense that all its parameters (choice of neighborhood and search heuristics, and the parameters of these heuristics) are learned dynamically. This is done by maintaining weights over the likelihood of choosing each heuristics and over a vector of possible values for the heuristics parameters. The heuristics to be used and its parameters are then randomly chosen according to this distribution. Af-

ter each reoptimization, the effect of the heuristics is measured and the weights are updated: if a heuristics resulted in improving the solution, the probability that it is used again in the future will be increased, and vice versa.

An overview of LNS is presented on Figure 2.3.

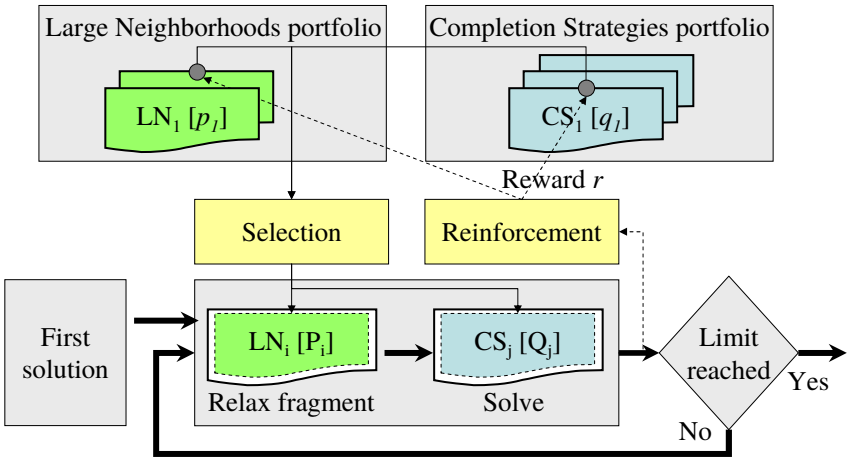


Figure 2.3: Self-Adapting LNS overview (from [LG07])

We can already see with this simple description that CP Optimizer relies on concepts which go far beyond "classical" CP: propagation remains a core concept, but the default search uses local search as much as tree search, this local search can itself be guided by a linear relaxation of the problem, learning is used at all stages, etc.

2.5.3.3 Genetic Algorithm

The Genetic Algorithm search method is a multipoint local search algorithm (it works with a population of solutions). Since enumerating the time is extremely costly and the solution space is typically not convex, the local search should not be performed directly on the decision variables of the problem (which include starting and ending times), but should instead be performed on an indirect representation of the problem, which is then decoded into an assignement of values to the decision variables.

This is what is done in CP Optimizer's genetic algorithm: a genome is maintained and evolved, which encodes mostly precedences between activities. It is then interpreted through a decoding step. In this step, the genome is transformed into additional constraints. A solution is then found using a completion algorithm (the parameters of the completion algorithm are also evolved and decoded from the genome).

This algorithm is generally not as robust as the Large Neighborhood Search, but has better performances in some of the benchmark instances and is thus frequently useful on problems with a complex solution landscape. This is mostly due to better diversification.

In addition, it is often used in conjunction with other search methods such as LNS to try and improve the solution they found.

2.5.3.4 Failure Directed Search

The Failure Directed Search is used when we aim at improving the dual bound on the current solution and hopefully prove that this solution is optimal. It is started under the conditions that the search space is small enough, since FDS is RAM-intensive, and that the LNS is not making progress anymore. In this case indeed, we assume that there is probably no better solution and start working towards a proof. If a better solution exists it is very hard to find with the common techniques and we need to cover the full search space anyway.

In order to cover all of the search space rapidly, we now aim at finding failures as quickly as possible. Moreover, no-goods are learned when failures are found so as to avoid exploring the same part of the search space in the future and further reduce the search space.

In order to direct the search towards failures, the decisions (branching variable and value) are rated: those which lead to stronger domain reduction, and ideally failure, are preferred and will be used with a greater propability during the next restarts.

More information about this technique can be found in [VLS15].

2.5.4 Propagations

There are several keys to implementing propagations efficiently in a constraint programming engine.

A first good practice is to use a propagation queue, running the least expensive (in terms of computing time) propagators in priority over the heavier ones. In particular, once a heavy propagator manages to change a domain, it should be interrupted in favor of the lighter propagators, instead of trying to reach the fixed point of that propagation. The fixed point of all propagations though must be reached before branching.

The goal of incrementality is to maintain information between the nodes during the search, in order to avoid trying the same propagations several times and to restudy only the part of the propagations which may have been affected by the last branching decision. The domain values modified by the last branching decision are called delta domain in this context. A propagation can also be

made incremental with the use of supports, which are logical conditions whose truth value must change before it is necessary to reexamine this propagation.

Typically, several propagation algorithms are implemented for each constraint. These algorithms have varying properties in terms of algorithmic complexity, deduction power, incrementality, cooperation with other factors such as the presence of other constraints or the search strategy being used.

Trade-offs have thus to be considered when choosing a propagation algorithm in a given situation.

An instance of such a trade-off is between the time spent propagating and the propagation power. Typically, if we want to search for a feasible solution (that is, improve the primal bound), we will use a search algorithm which will quickly sample the search space in different regions. In this case, we want to move quickly through the search space and need relatively inexpensive propagation algorithms (typically, no more than $O(n \log n)$ where n is the number of tasks), at the expense of the propagation power, with the hope that we will not end up in too many dead-ends. On the other hand, if we want to improve the dual bound, we need to cover the search space more extensively, and will benefit from longer but more powerful propagations, which will significantly reduce the size of the search space. For example, the Edge Finder (see below in Subsection 3.4.3), which is an expensive propagation, does not combine well with the LNS when trying to find good feasible solutions, but it works very well in combination with a search strategy aimed at improving the dual bound, such as FDS.

Another trade-off concerns the incrementality properties of a propagator. For example, the Timetable propagation can be updated in linear time from a node to one of its successors, thus making it the cornerstone of propagators for the cumulative constraint, in spite of the existence of much stronger propagators, but with poorer incrementality properties.

In the next chapter, we will detail some propagations which are used in constraint-based scheduling, notably on the cumulative constraint.

Chapter 3

The Cumulative constraint

We introduce in this chapter the cumulative constraint, briefly mentioned previously, which is at the heart of this thesis. As we will see, this constraint is used to model very diverse situations and is widely used, since we find a cumulative aspect in most scheduling problems. Besides, there exist several families of powerful algorithms to deal with this constraint.

We begin in Section 3.1 by defining this global constraint and the notation which we will use further on. We continue in Section 3.2 by introducing the Resource Constrained Project Scheduling Problem, a fundamental problem in scheduling, which has many uses in and of itself, and has many close ties with the cumulative constraint. Then in Section 3.3, we present numerous varied examples of industrial applications of the cumulative constraint which motivates the extensive study of this constraint on top of its theoretical interest. Finally in Sections 3.4 to 3.8, we present different algorithmic frameworks to solve cumulative schedules.

3.1 Definition and notations

We focus in this chapter on the discrete cumulative resource, an abstraction for the assignment of a fixed quantity of a resource to a task during its execution period. At any point in time, the total resource usage by the tasks being run must not exceed the total resource capacity. This abstraction is heavily used in constraint-based scheduling to model the allocation of a limited resource, such as manpower, budget, parallel machines, reservoirs. In this context, we talk of the cumulative *constraint*.

Definition 1. Given a discrete cumulative resource of capacity C on a set of n tasks with respective length p_i and demand c_i on the resource, we say that the resource can be satisfied if and only if for each task there exists a start time t_i

such that:

$$\forall \text{ time } t, \quad \sum_{\substack{i \in [1,n] \\ t_i \leq t < t_i + p_i}} c_i \leq C$$

In this case, $t_1, ..., t_n$ is called a *valid schedule*.

An example of a common application of this constraint is in the famous Resource Constrained Project Scheduling Problem: this problem can be seen as minimizing the makespan of a set of non-preemptive tasks of given lengths, under a cumulative constraint such as we just introduced, in addition to precedence constraints between tasks.
The cumulative constraint was introduced as a constraint in the Constraint Programming framework in [AB93].

Note that this problem is related but clearly different from the two-dimensional packing problems [LMM02], such as strip-packing. One can obtain excellent relaxations of two-dimensional packing problems with cumulative resources though, and all the techniques developed for the cumulative resource apply to two-dimensional packing as well.

We will see later in this chapter several algorithmic techniques to deal with the cumulative constraint, but a fundamental tool, used by several of these techniques, is the notion of *energy* of a set of tasks over a cumulative resource:

Definition 2. Given a discrete cumulative resource of capacity C and a set of n tasks with respective lengths p_i and demands c_i, the *energy bound* of the tasks on the resource is $E = \sum_{i=1}^{n} \frac{p_i c_i}{C}$.

Notice that the energy gives a lower bound for the total time needed to run all the tasks.

For example (see Figure 3.1), on a resource of capacity $C = 4$, if we have $n = 5$ tasks of lengths $p = (3, 4, 9, 1, 4)$ and respective demands $c = (4, 3, 1, 2, 1)$, then the energy lower bound on the makespan is $\frac{3\times4+4\times3+9\times1+1\times2+4\times1}{4} = 9.75$ (the actual minimum makespan for this example is 12).

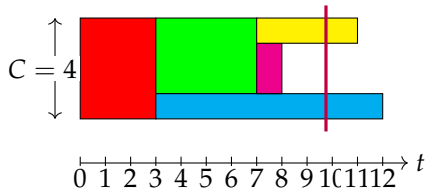


Figure 3.1: Example instance. Energy bound: 9.75.

3.2 RCPSP

The cumulative constraint is often studied in the context of the Resource Constrained Project Scheduling Problem (RCPSP), as these two notions are intimately linked. The RCPSP is indeed the scheduling problem containing only cumulative constraints (possibly several of them) and precedence constraints between the tasks.

The reason for combining the study of these two families of constraints is that they act in an orthogonal fashion on the solution: intuitively, the precedence constraints define the structure of the solution along the temporal axis, while the cumulative constraints act on the behaviour of the problem at a given point in time. Thus, they form the prototypes of constraints acting on these two dimensions and enable the study of their interactions, which have yielded rich and fruitful algorithmic results.

In addition to its theoretical interest in relation with the cumulative constraint, the RCPSP, in spite of its simplicity, is also applicable to a broad variety of practical problems. Indeed, it generalizes many practical production problems, such as parallel machines scheduling and shop problems, as we will see below.

Of course, even though the cumulative resource is often presented and studied in the context of the RCPSP, it is much more general and can be used in many different settings. This will be illustrated on different examples in the following section.

There exists an extensive body of literature about the RCPSP: several hundreds of articles and a dozen of books have been published on this topic. We will only cite one of them here: the recent and excellent [ADN13] which contains surveys of all the sub-fields of the study of the RCPSP and references for the interested reader.

3.2.1 Formal definition

Let us now give a formal definition of the RCPSP. Given:

- R cumulative resources $r = 1, \ldots, R$ of respective capacities C_r.
- n activities $i = 1, \ldots, n$ of respective durations p_i and demands $d_{i,r}$ on each resource r.
- and a set of precedences $i \to j$ such that the directed graph whose nodes are the activities and edges are the precedences is acyclic.

we must find a start date S_i for each activity such that:

- at any point in time, for each cumulative resource, the sum of the demands of the tasks being executed does not exceed the capacity of that resource:

$$\forall \text{time } t, \forall r \in [1, R], \quad \sum_{\substack{i \in [1,n] \\ S_i \leq t < S_i + p_i}} d_{i,r} \leq C_r.$$

- for each precedence $i \rightarrow j$, the inequality $S_i + p_i \leq S_j$ is satisfied, so that activity i finishes before activity j begins. Preemption is not allowed.

- the makespan $\max_{i \in [1,n]} (S_i + p_i)$, that is the latest completion time of any activity, is minimized among all the schedules which satisfy the previous two constraints. Other objectives are possible in variants of the RCPSP.

Without loss of generality, all quantities C_r, $d_{i,r}$ and p_i are assumed to be integer.

3.2.2 Example

We show here a small example of a project with $n = 5$ activities and $R = 2$ cumulative resources of respective capacities $C_1 = 3$ and $C_2 = 5$. For each activity i, the lengths p_i and demands $d_{i,1}$ and $d_{i,2}$ on the two resources are given in the following table:

i	1	2	3	4	5
p_i	5	3	3	4	4
$d_{i,1}$	2	2	1	1	1
$d_{i,2}$	1	2	4	3	1

Moreover, there are two precedences in this instance, $1 \rightarrow 2$ and $3 \rightarrow 4$, as shown on the precedence graph, represented on Figure 3.2.

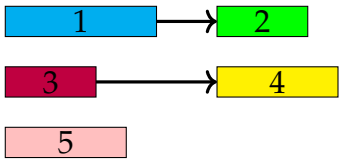


Figure 3.2: Precedence graph

A possible optimal schedule for this instance, of makespan 10, is given by the following start dates:

i	1	2	3	4	5
S_i	0	7	0	5	3

The consumption on the two cumulative resources is represented on Figure 3.3.

(a) Resource 1 (b) Resource 2

Figure 3.3: An optimal schedule

3.2.3 A versatile model

The RCPSP, in spite of its simplicity, is a surprisingly expressive model. For example, the problems we introduced above in Section 2.1, such as single machine problems, parallel machine problems and shop scheduling problems, can be modeled as RCPSPs. We give here an outline of these reductions.

Single-machine problems, sometimes also called disjunctive problems, where activities have to be scheduled on a resource which accepts one activity at a time, are easily modeled with a cumulative resource of capacity $C_1 = 1$, on which each activity has a demand $d_{i,1} = 1$.

Similarly, parallel identical machines problems, where jobs can be scheduled on m identical machines with the same processing speed, can be modeled with an RCPSP with one cumulative resource of capacity $C_1 = m$ and each activity has a demand $d_{i,1} = 1$.

The open shop scheduling problem with n jobs and m machines can be modeled with an RCPSP with $n + m$ cumulative resources, all of capacity $C_i = 1$. The first n resources correspond to the jobs, and the following m resources correspond to the machines. The activities correspond to the operations $O_{i,j}$, with a demand of 1 on resource j corresponding to job J_i, and a demand of 1 on resource $n + \mu_{i,j}$, corresponding to the use of the machine. The special cases of the shop scheduling problems we mentioned above can also be modeled as RCPSPs.

We refer the reader to Chapter 1 of [BK12] for more details on modeling with the RCPSP.

3.2.4 Algorithmic complexity

In spite of its practical relevance, this problem is of formidable difficulty: it is strongly NP-hard, even without precedence constraints and only one cumulative resource of capacity 3 [GJ75]. The reduction from 3-PARTITION is intuitive.

Contrary to many other NP-hard problems, it is also poorly tractable in practice, even for instances of modest sizes. Surprisingly, small instances which contain only 60 tasks and are more than 20 years old as this thesis is written are still unsolved to optimality to this day, in spite of the tremendous recent improvement in algorithms and computing power. The most commonly used benchmark instances for the RCPSP have been collected in the PSPLIB [KS97].

3.2.5 OPL model

A model for the RCPSP in the OPL language is presend in Listing 3.1. Let us detail this code:

Lines 1-12 correspond to the data input. The RCPSP instance has n tasks and R resources.

A data structure containing the informations of each task (length, demand on each of the R cumulative resources, and direct successors in a precedence constraint) is initialized lines 4-9 and filled line 11.

The capacity of each of the R cumulative resources is set line 12.

Line 14, the decision variables of the model are declared. They are here of type `interval` which means that they represent *interval variables*, that is time period over which the corresponding task will be executed. The start and the end of that period are to be determined by the solver.

Lines 16-18, the cumulative functions `usage[r]` are declared. Their value is defined as a sum over the tasks of *pulses*, that is functions of the time whose value is the demand of the task on that cumulative resource on the execution period of the task and null elsewhere. In this way, the value of the cumulative function at any point in time is the sum of the demands of the tasks being executed.

The objective function of the model is declared line 20. It states the *makespan* objective, that is the minimization of the latest end time of any of the tasks. While widely regarded as quite academic, the makespan is a sensible measure of the throughput of a production system.

The constraints of the model appear in the ‘‘`subject to`’’ block, lines 21-26. As we mentioned above, there are two families of constraints:

Lines 22-23 express the cumulative constraints: for every cumulative resource, the corresponding cumulative function must never exceed the capacity

Listing 3.1: RCPSP model

```
1  int R = ...; //number of resources
2
3  tuple Task {
4    key int id;
5    int length;
6    int demands[0..R-1];
7    {int} succs;
8  }
9
10 {Task} Tasks = ...; //characteristics of tasks
11 int Capacity[r in 1..R] = ...; //resources capacities
12
13 dvar interval intervals[task in Tasks] size task.length;
14
15 cumulFunction usage[r in 1..R] =
16   sum (task in Tasks: task.demands[r]>0)
17     pulse(intervals[task], task.demands[r]);
18
19 minimize max(t in Tasks) endOf(intervals[t]);
20 subject to {
21   forall(r in 0..R-1)
22     usage[r] <= Capacity[r];
23   forall(task in Tasks, succId in task.succs)
24     endBeforeStart(itvs[task], itvs[<succId>]);
25 }
```

of the cumulative resource.

Lines 24-25 correspond to the precedence constraints: for all declared direct successors to a task, we declare an `endBeforeStart` constraint, which states a precedence betwen the two intervals given as parameters.

3.2.6 RCPSP with Multiple Modes

An important variant of the RCPSP, which we do not deal with in this thesis but should be mentioned for completeness, is the RCPSP with Multiple Modes (RCPSPMM).

In this variant, a set of modes M_i is associated with each activity i. To each mode $m \in M_i$ corresponds a different processing time $p_{i,m}$ and different de-

mands $d_{i,r,m}$ on the cumulative resources. This is used to model the frequent situations in which different processes or machines can be used to obtain the same result, albeit with a different processing time. In addition to setting starting dates, the solver must select a mode for each activity.

In CP Optimizer, *alternatives* can be used to model the RCPSPMM.

3.3 Examples of industrial applications of the cumulative constraint

Since this thesis was prepared in an industrial setting, we wish to present some concrete examples of the usage of the cumulative constraint, to show how versatile it is. In our experience, more than half of all scheduling models involve a cumulative constraint. Our goal is to represent the variety of its uses, both from an application and a modelization standpoint.

3.3.1 Exclusive zones

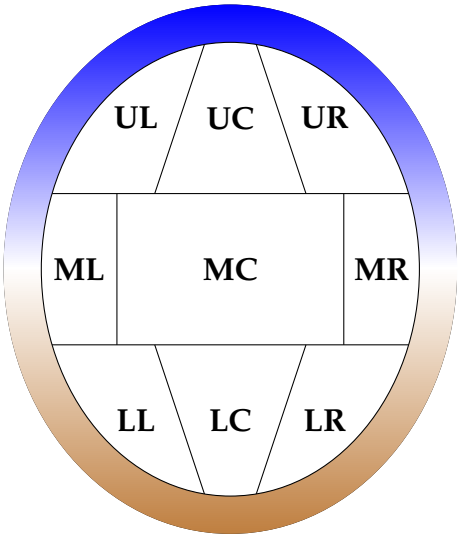
Many areas in an aircraft under construction are cramped and can only accomodate a limited number of workers, or are supported by scaffolding and have severe weight limits. Exclusive zones are extensively used in aircraft manufacturing to account for these restrictions. The principle is to divide the aircraft into zones, and to limit the number of people working simultaneously in a zone (or a union of zones) using a cumulative constraint.

Figure 3.4 shows an example of splitting the forward section of an aircraft into different exclusive zones, and the associated constraints.

The cumulative resource is often used to model such volume limitations in the case of processing several items at once, for example in chemical reactors or in ovens, in semiconductor manufacturing.

3.3.2 Workforce scheduling

A common application of scheduling is the building of rosters. Different tasks have to be assigned to different employees. Each task requires a certain skill, and employees each have a set of skills. The easiest way of modeling this with CP Optimizer is to have an interval for each task t_i, and to create optional tasks t_{ij} for each task t_i and each employee j which has the skills to complete task t_i. Two types of constraints are then used: an alternative from each main task t_i to the subtasks t_{ij}, which ensures that the task will be assigned to exactly one employee, and a no-overlap constraint for each worker j on all the tasks t_{ij}



Zones	Max number of people
UL	1
UC	1
UR	1
ML	1
MC	2
MR	1
LL	1
LC	2
LR	1
UL+UC+UR	2
ML+MC	2
MC+MR	2
UL+UC+UR+LL+LC+LR	3
LL+LC+LR	2

Figure 3.4: Exclusive zones in the forward section of an aircraft

potentially assigned to that worker. This model is shown on Figure 3.5 and in Listing 3.2.

It is very helpful here to add a redundant cumulative constraint for each skill. These cumulative constraints will relax the exact assignment of tasks to workers, but they will have a more global view of all the tasks sharing certain workers. This will enable them to make deductions that the no-overlap constraints would have missed. For each skill, we thus define a new cumulative constraint. Its capacity is the number of workers having that skill, and the de-

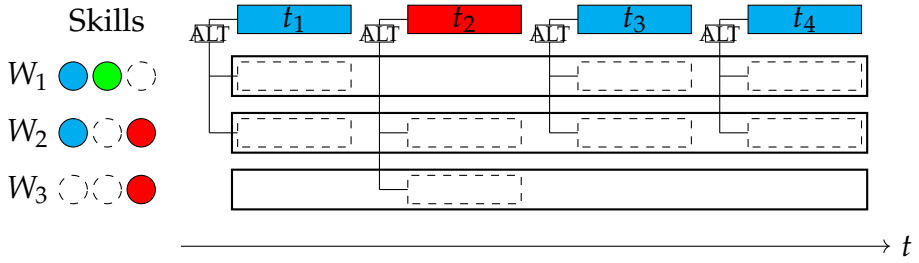


Figure 3.5: Workforce scheduling

mand of each task t_i on the constraint is 1 if it requires that skill, 0 otherwise, as shown in Listing 3.3.

Using a cumulative resource as an approximation for identical machines is often a wise and efficient practice for non-critical parts of a model.

3.3.3 Long-term capacity planning

In supply chain management, capacity planning consists in matching the resource availabilities and customer demands for the products of a company. In general, the long term strategy of the company is at stake.

This problem is generally modeled as a linear problem by assimilating the production as a cumul of demands over a small number of fixed time buckets. The expected demands come from strategic forecasts, the time horizon is huge, typically in years, and the buckets duration is long as well, typically one month. The objective is then to choose which resources to use for the production, and to compute an approximate production date.

Nevertheless, the long duration time buckets relaxation is not precise enough for some cases. This is for example the case when precedence constraints between demands, or costs in case of early or late delivery, are key factors. It prevents from safely aggregating the demands in buckets and requires a continuous time definition. If we suppose that the relaxation of the production system as cumulative resources is still valid, the core problem is then an RCPSP (eventually an RCPSPMM) with a huge number of tasks, typically up to one million, and a large set of cumulative resources, typically in the hundreds.

3.3.4 Berth allocation

This example is an original, though not uncommon, application of the cumulative constraint to a geometric problem. A commercial port has a linear quay of length L. Ships can dock anywhere along the quay, but their berthing locations obviously cannot overlap, see Figure 3.6. The ships are indexed from

Listing 3.2: Workforce scheduling model

```
int n = ...; //number of tasks
int w = ...; //number of workers
int s = ...; //number of skills

int taskType[i in 1..n] = ...; //skill needed by a task
int skillsWorker[1..w][1..s] = ...; //1 if worker has
   the skill, 0 otherwise
int skillAvailability[k in 1..s] = sum(j in 1..w)
   skillsWorker[j][s];

dvar interval tasks[1..n];
dvar interval assignedTasks[1..n][1..w] optional;

constraints {
forall(i in 1..n) {
  alternative(tasks[i],all(j in 1..w: skillsWorker[j][
     taskType[i]] == 1) assignedTasks[i][j]);
  forall(j in 1..w: skillsWorker[j][taskType[i]] == 0)
    !presenceOf(assignedTasks[i][j]);
}
forall(j in 1..w)
  noOverlap(all(i in 1..n) assignedTasks[i][j]);
}
forall(k in 1..s: skillAvailability[k]>0)
  sum(i in 1..n: taskType[i] == k) pulse(tasks[i], 1) <=
     skillAvailability[k];
```

1 to n. Ship i has a length of l_i meters. Its arrival time at the port is a_i and it must depart before time d_i while spending at least s_i time units docked. The goal is to find berthing locations and times which prevent the overlaps.

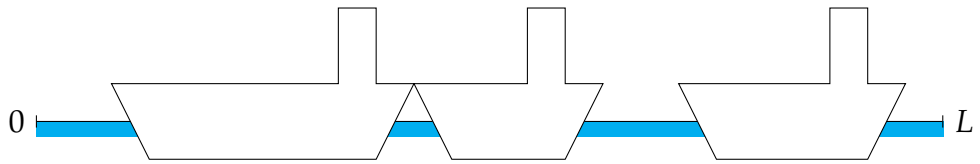


Figure 3.6: Berth allocation

This problem is a strip-packing problem, if we think of the temporal dimen-

Listing 3.3: Redundant cumulative

```
int skillAvailability[k in 1..s] = sum(j in 1..w)
   skillsWorker[j][s];

constraints {
forall(k in 1..s)
  sum(i in 1..n: taskType[i]==k) pulse(tasks[i],1) <=
     skillAvailability[k];
}
```

sion as the infinite dimension in a strip-packing problem. Since the cumulative constraint is a very strong relaxation of bidimensional packing problems, all the more so when there is a "thin" dimension, we can get a solution to the original problem which is almost always feasible just by modeling it as a cumulative problem, as shown in Listing 3.4.

Listing 3.4: Berth allocation model

```
int n = ...; //number of docking events
int L = ...; //length of the quay

dvar interval ship[i in 1..n] ...;
cumulFunction quayOccupation = sum(i in 1..n) pulse(ship
   [i], l[i]);

constraints {
  forall(i in 1..n) {
    startOf(ship[i]) >= a[i];
    endOf(ship[i]) <= d[i];
    lengthOf(ship[i]) >= s[i];
  }
  quayOccupation <= L;
}
```

In many instances and for all practical purposes, since ships have a large enough length and only a small number of them can fit on the quay at the same time in the real-world problem, this model provides an exact relaxation of the original problem.

The principle of using a cumulative resource to model a strip packing problem with a thin dimension applies to vehicle loading problems such as truck or train loading. The idea consists in using the temporal dimension of the cumulative resource to model the length of the vehicle, as shown on Figure 3.7.

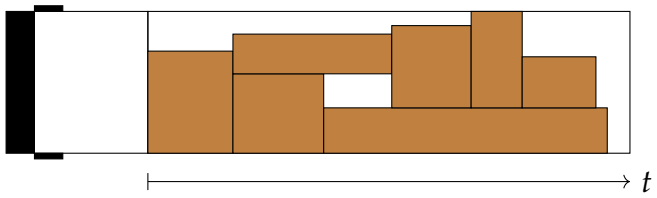


Figure 3.7: Truck loading

3.3.5 Balancing production load

In CP Optimizer, the demand of a task can also be a decision variable, and not only a data of the problem. One example of a use case is in balancing a production load.

In many industrial problems, the real objective is to balance the production as evenly as possible, which is not easy to express as an objective function in an optimization model. A way of accomplishing this is to minimize the peak utilization of the resources. In this example, a dummy task of variable demand D is added to the cumulative constraint and the objective is to maximize D, as shown in Figure 3.8 and Listing 3.5.

Many variants of this idea can be used to balance a load, such as having a dummy task per work shift.

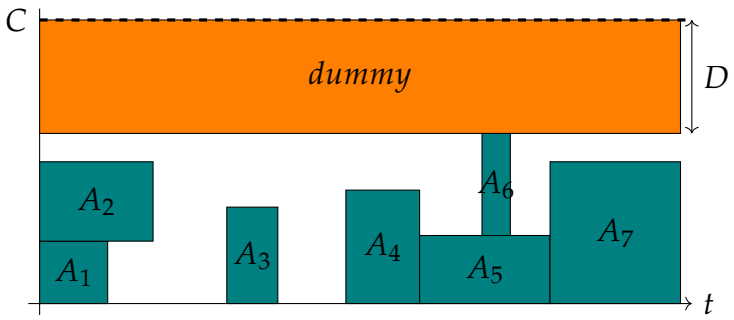


Figure 3.8: Balancing production load

Listing 3.5: Balancing production load

```
int n = ...; //number of tasks
int C = ...; //cumulative capacity

dvar interval A[i in 1..n] ...;
dvar interval dummy ...;
cumulFunction f = pulse(dummy, 0, C) + sum(i in 1..n)
   pulse(A[i], 1);

maximize heightAtStart(dummy, f);
constraints {
  f <= C;
}
```

3.3.6 Batching

Another type of constraint, related to the cumulative constraint as defined previously, is the *alwaysIn* constraint. We do not deal with the algorithmic aspects of the alwaysIn constraint in this thesis, but mention it here for comprehensiveness. $alwaysIn(f, a, h_{min}, h_{max})$ states that during the execution interval of activity a, the cumulative function f, representing the sum of the demands of the tasks being executed at a certain point in time, must always be at a level in the range $[h_{min}, h_{max}]$.

In the following example which is sometimes encountered in process scheduling models (OPL code in Listing 3.6 and illustration in Figure 3.9), the alwaysIn constraint is used to locate production periods, in order to fit maintenance tasks between those periods. The production tasks take place during the A_i intervals. We want a W_j interval to be present exactly whenever an A_i interval is present. To this end, the constraint `alwaysIn(CW, A[i], 1, 1)` ensures that exactly one W_j interval is present whenever an A_i interval is present. The constraint `alwaysIn(CA, W[i], 1, n)` ensures that at least one A_i interval is present whenever a W_j interval is present, so that no W_j interval is present when no A_i interval is present.

A natural extension of the cumulative resource is the "reservoir", which is a succession both of filling and emptying events, for which we have a minimum level constraint (in general 0) and a maximum level constraint (the reservoir volume).

In this thesis, we will not deal with negative demands nor minimum levels in cumulative resources.

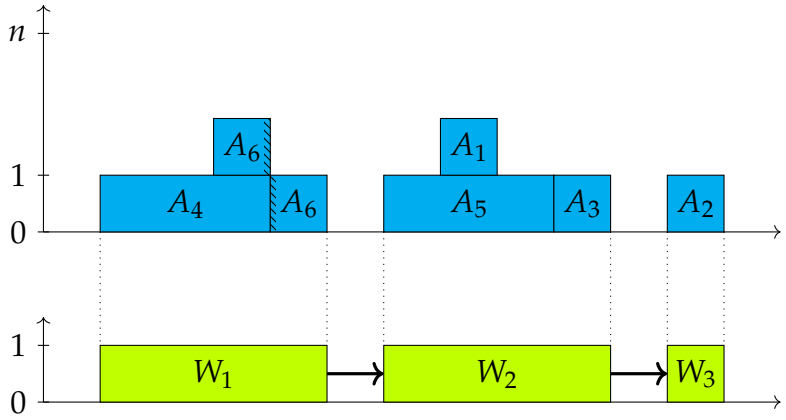


Figure 3.9: Intervals synchronization

Listing 3.6: Intervals synchronization

```
int n = ...;

dvar interval A[i in 1..n] ...;
cumulFunction CA = sum(i in 1..n) pulse(A[i], 1);
dvar interval W[i in 1..n] optional in 1..horizon;
cumulFunction CW = sum(i in 1..n) pulse(W[i], 1);

constraints {
  forall(i in 1..n) {
    alwaysIn(CW, A[i], 1, 1);
    alwaysIn(CA, W[i], 1, n);
  }
  forall(i in 1..n-1) {
    endBeforeStart(W[i], W[i+1]);
  }
}
```

3.4 Cumulative propagation

We describe in this section the most common propagations for the Cumulative constraint. These propagations only take into account the current bounds for the start and end times of the different activities, except for the Precedence Energy which also takes precedence relationships into account. Most of these propagations rely on the notion of Energy Bound, defined above in Section 3.1.

The general principle is to select a set of activities, select a time interval where these activities should be scheduled, and move an activity if the total energy of the activities exceeds the available energy in the window.

There are dominance relationships between these propagations, in the sense that the deductions made by some of them are subsubmed by the deductions made by another one with a higher algorithmic complexity. In particular, Energy Reasoning is stronger than the Edge-Finders. It is also stronger than the Timetable. It is worth noting though that Energy Reasoning and Not-First, Not-Last are incomparable and can both make deductions that the other cannot.

3.4.1 Timetable

The timetable propagation was introduced in [AB93]. A lot of improvement has since been made on this topic and exists unpublished in the community. The timetable has a quadratic worst-case complexity but its amortized complexity is linear. In spite of its relative propagation weakness, its low complexity makes it the basis of propagating the cumulative constraint.

The timetable amounts to maintaining arc-B-consistency on the equation from Definition 1 which defines a satisfied cumulative resource.

The principle is to maintain a so-called resource histogram, recording for each point in time the minimum amount of resource consumption: first we compute for each task its compulsory part, that is

$$M_i(t) \triangleq \begin{cases} c_i & \text{if } lst_i \leq t < eet_i \\ 0 & \text{otherwise} \end{cases}$$

We accumulate for each time point the compulsory parts of all tasks:

$$U(t) \triangleq \sum_i M_i(t) = \sum_{i \in [\![1,n]\!] : lst_i \leq t < eet_i} c_i$$

Several deductions can be made with this histogram:

- If $\exists t / U(t) > C$, then we know that the current problem has no solution and the search must backtrack.
- If for a particular task A_i, $\exists t_0 / \max(lst_i, eet_i) \leq t_0 < let_i$, $U(t_0) - M_i(t_0) + c_i > C$, then A_i cannot end after t_0, otherwise the resource would be over-consumed. let_i can then be updated to t_0. A symmetric rule exists for updating the start times.

In addition to its low complexity, a major advantage of the timetable is that it is easily extendable. It can easily support extensions such as reservoirs (negative demands and minimum level constraints on a cumulative), as well as the joint propagation of other resources such as state functions with transition time, without incurring a performance penalty.

The implementation in CP Optimizer is fully incremental with respect to changes to the interval variables domains.

3.4.2 Disjunctive

If the sum of the demands of two activities A_i and A_j exceeds the capacity of the cumulative, they cannot overlap in time, so one of them must precede the other. The disjunctive constraint maintains arc-consistency on the formula:

$$\left[c_i + c_j \leq C\right] \vee \left[A_i \rightarrow A_j\right] \vee \left[A_j \rightarrow A_i\right]$$

This constraint can be propagated on all activities with a $O(n \log n)$ complexity, as explained in [Vil04]. This propagation is surprisingly powerful in practice, since many instances of cumulative problems exhibits tasks which cannot overlap.

3.4.3 Edge Finding and derivatives

The Edge-Finding family of propagation techniques reason about the order of execution of activities. Specifically, these algorithms determine if a given activity A_i must execute before (or after) a set of activities Ω. Two types of informations can be drawn from this: new precedence relations, sometimes called implicit precedences or "edges", and more importantly new bounds on the execution dates of activities.

The general principle is to check if the total energy of a set of activities Ω fits in the available energy for Ω, otherwise a conflict is detected:

$$C(let_\Omega - est_\Omega) < e_\Omega \Rightarrow \text{fail}.$$

There exists a breadth of Edge-Finder variants, with different propagation power versus algorithmic complexity trade-offs, and research is very active in this area. We present the most common versions.

Edge-Finder The original cumulative Edge-Finder was introduced in [Nui94]. For $A_i \notin \Omega$, if

$$C(let_\Omega - est_{\Omega \cup \{A_i\}}) < e_{\Omega \cup \{A_i\}},$$

then A_i must end after lct_Ω. Indeed, the previous condition is satisfied if the energy overflows when A_i is forced to finish no later than let_Ω.

The original complexity of this propagation was $O(n^2)$ where n is the number of activities, and it was improved in [Vil09a] to $O(kn \log n)$, where k is the number of different demands of the activities, thanks to a new data structure called Θ-tree.

Extended Edge-Finder The Extended Edge-Finder was also introduced in [Nui94]. Instead of considering the $[est_{\Omega\cup\{A_i\}}, let_\Omega)$ window as previously, only the window $[est_\Omega, let_\Omega)$ is now considered and only the part of the energy of A_i which necessarily falls into this window is taken into account.

The full propagation condition is thus, for $A_i \notin \Omega$ and $est_i \leq est_\Omega < eet_i$, if

$$C(let_\Omega - est_\Omega) < e_\Omega + c_i(eet_i - est_\Omega),$$

then A_i must end after lct_Ω.

The original complexity of this propagation was $O(n^3)$, and it was improved in [MVH08] to $O(kn^2)$.

Timetable Edge-Finder The Timetable Edge-Finder was introduced in [Vil11], and combines the Timetable propagation with the Edge-Finder, by also taking into account the timetable in the overflow condition.

We defined above in Subsection 3.4.1 the $U(t)$ function, which sums the mandatory energy of all tasks at time t.

We can thus improve the overload checking rule by adding the energy from the timetable:

$$C(let_\Omega - est_\Omega) < e_\Omega + \sum_{t=est_\Omega}^{let_\Omega} U(t) \Rightarrow \text{fail}.$$

Depending on the relative positions of A_i and Ω, we can add to this equation the amount of energy of A_i that necessarily falls in the $[est_\Omega, let_\Omega)$ window and is not yet taken into account in the timetable to obtain an adjustment rule. The complexity of this propagation is $O(n^2)$.

3.4.4 Not-First, Not-Last

The Not-First, Not-Last propagation was originally introduced in [Nui94]. It gives necessary conditions under which an activity A_i has to be scheduled after at least one activity (not-first) or before at least one activity (not-last) of a set Ω of activities. This is complementary to edge-finding.

We now describe the Not-First rule. The Not-Last rule is similar.
For a set of activities Ω and an activity $A_i \notin \Omega$, if

$$est_\Omega \leq est_i < \min_{j\in\Omega} eet_j, \text{ and}$$

$$e_\Omega + c_i(\min(eet_i, let_\Omega) - est_\Omega) > C(let_\Omega - est_\Omega),$$

then A_i must start after $\min_{j\in\Omega} eet_j$ and est_i can be updated.

Indeed, if we force activity A_i to start at its earliest start time, then no activity of Ω can be completed before A_i is started, since $est_i < \min_{j\in\Omega} eet_j$. Thus an energy $c_i(est_i - est_\Omega)$ is occupied by no activity before A_i starts, as shown in green on Figure 3.10.

The sum of the energy of the activities in $\Omega \cup \{A_i\}$ on the window $[est_\Omega, let_\Omega)$ is thus $e_\Omega + c_i(min(eet_i, let_\Omega) - est_i) + c_i(est_i - est_\Omega)$, which overflows the available energy.

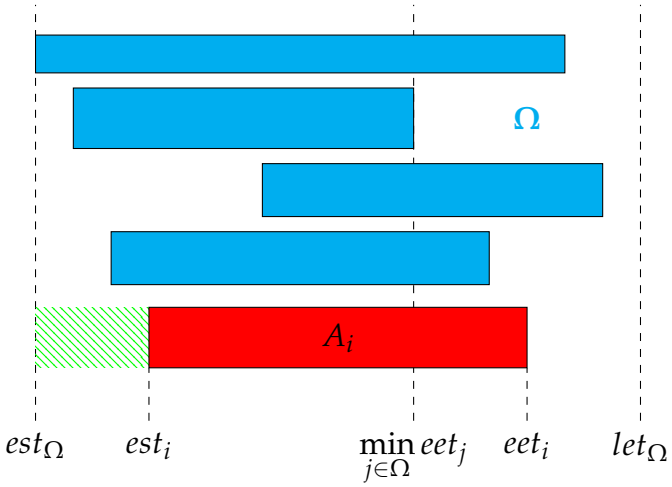


Figure 3.10: Not-First, Not-Last

We can thus deduce that:

$$est_i \geq \max_{\Omega\subseteq\{A_1,\ldots,A_n\}/\begin{cases} A_i \notin \Omega \\ est_\Omega \leq est_i < \min_{j\in\Omega} eet_j \\ e_\Omega + c_i(\min(eet_i, let_\Omega) - est_\Omega) > C(let_\Omega - est_\Omega) \end{cases}} \min_{j\in\Omega} eet_j$$

The original complexity of propagating this condition on all tasks was $O(n^3)$. It was improved to $O(n^2 \log n)$ in [SW10].

3.4.5 Energy Reasoning

The energy reasoning was originally introduced in [EL90]. It consists in fixing a time window, and accumulating for all the the tasks the minimum energy within the time window. The propagation is triggered when an overflow is detected. This propagation is extremely powerful and subsumes several other propagations (notably timetabling and edge-finding). But its original computational complexity of $O(n^3)$ made it unusable in practice. [EL90, BPN01].

The energy reasoning has seen much interest and several improvements in the recent years, notably in a series of two articles [DP13, DP14] in which the constant factor of its complexity was divided by seven, and in techniques aiming at using machine learning to predict cases when running the expensive $O(n^3)$ Energy Reasoning propagator will be beneficial, so as to use this propagation sparingly [VCLSo14].

As part of this doctoral work, we developed a new propagation algorithm with a complexity of $O(n^2 \log n)$ [Bon16], making this propagation useful in a higher number of cases.

We do not detail this propagation here, since it will be extensively covered in Chapter 5.

3.4.6 Energy Precedence

The energy precedence propagation, introduced in [Lab03a], is different from the previous propagations in that it combines information coming from the temporal network (precedence constraints) with the information on the cumulative constraint and the current domains of the intervals. This propagation is especially useful when used in combination with a precedence-based search (a branching scheme which adds precedence constraints to the model).

The energy precedence ensures that the cumulative resource provides enough energy for all the predecessors of a task to fit before that task can start. More formally, for each subset Ω of predecessors of a task A_i, the earliest start time of a task in Ω plus a lower bound on the running time of all the tasks in Ω provides a lower bound on the earliest start time of A_i:

$$\forall \Omega \subseteq \{j \in [\![1, n]\!] : A_j \to A_i\},\ est_i \geq est_\Omega + \left\lceil \frac{e_\Omega}{C} \right\rceil$$

There exists a symmetric rule with the successors and latest end time of i.

Since subsets Ω of the form $\{j \in [\![1, n]\!] : A_j \to A_i, est_j \geq t\}$ dominate the other subsets for this propagation, and there are only p of them where p is the maximal number of predecessors of a given activity in the temporal network ($p < n$), the energy precedence can be propagated with a worst-time complexity $O(n(p + \log n))$.

3.5 MIP formulations

Numerous MIP formulations have been proposed for the RCPSP. Nevertheless, a major difficulty in coming up with good formulations is that the cumulative constraint is deeply non-linear with respect to the dates of the activities. In other words, it is very difficult to express the temporal and cumulative constraints jointly using linear inequalities.

This requires compromises on the size of the formulation and the quality of the linear relaxation, which are done by choosing different variables, representing more or less directly the temporal and cumulative constraints, and thus by different linearizations, more or less precise and more or less compact. Nevertheless, there is no good compact linearization and we have to choose between either a weak approximation or a pseudo-polynomial model. For this reason, only small instances have been solved to optimality, or instances with additional properties that could be exploited in the MIP model. Additionally, the linear relaxations of these formulations are generally weak.

The fundamental notion in most of these formulations and attempts at linearizing the cumulative constraint is that of antichains (set of activities which are incomparable in the precedence relation and can a priori be scheduled simultaneously).

Following Queyranne and Schulz's [QS94] classification of MIP formulations, we will group them hereafter by the type of decision variables used.

Time indexed formulations Time indexed formulations were introduced in [PWW69]. They are, perhaps surprisingly, the most commonly used MIP formulations for the RCPSP, owing to their simplicity, in spite of their poor practical performance on most instances. The basic principle (Discrete Time formulation) is to divide the scheduling horizon H into unit time slots and to use time-indexed binary variables: for any activity i and time $t \in [0, T]$, $x_{it} = 1$ if activity i starts at time t, 0 otherwise. With these variables, the precedence constraint from i to j is expressed as $\sum_{t=0}^{H} t x_{jt} - \sum_{t=0}^{H} t x_{it} \geq p_i$ and the cumulative constraint as $\sum_{i=1}^{n} c_i \sum_{\tau=t-p_i+1}^{t} x_{i\tau} \leq C$ for all times $t \in [0, H-1]$.

In the Disaggregated Discrete Time formulation, the precedence constraints are desagregated into $\sum_{\tau=0}^{t-p_i} x_{i\tau} - \sum_{\tau=0}^{t} x_{j\tau} \geq 0$ for all times $t \in [0, H-1]$ as introduced in [CAVT87]. We obtain a stronger continuous relaxation and better lower bounds. Moreover, if the cumulative constraints are dualized with a La-

grangian relaxation, the remaining constraint matrix is totally unimodular. This fact was used in [MSSU03] to solve the remaining problem with a maximum flow algorithm.

An even stronger linear relaxation can be obtained by replacing the cumulative constraint as expressed above by its Dantzig-Wolfe decomposition, using antichains. More specifically, we call $\mathcal{P}$ the set of antichains P. $\mathcal{P}_i$ is the set of antichains containing activity i. The decision variable y_{Pt} is equal to 1 if antichain P is being executed at time t, 0 otherwise. The cumulative constraint is now expressed as:

$$\forall i \in [1, n], \sum_{P \in \mathcal{P}_i} \sum_{t \in [0, H-1]} y_{Pt} = p_i,$$

so that enough antichains are available to execute all of activity i.

$$\forall t \in [0, H-1], \sum_{P \in \mathcal{P}} y_{Pt} \leq 1,$$

to schedule only one antichain per time slot.

$$\forall i \in [1, n], \forall t \in [0, H-1], x_{it} - \sum_{P \in \mathcal{P}_i} y_{Pt} - \sum_{P \in \mathcal{P}_i} y_{P,t-1} \geq 0,$$

to synchronize the start date of i with the first use of an element of $\mathcal{P}_i$.

This formulation was introduced in [MMRB98]. It was further improved with constraint propagation and specific cutting planes in [BK00, BD04].

All these time indexed formulations are pseudo-polynomial in H.

Continous time formulations These formulations are based on two families of variables: n continuous variables S_i representing the starting time of each activity and n^2 binary sequencing variables x_{ij} between all pairs of activities, representing precedences between activities: $x_{ij} = 1$ if activity j does not start before activity i is completed, and 0 otherwise.

The basic constraints of these formulations enforce the asymmetry and the triangle inequality of the precedence relation: $x_{ij} + x_{ji} \leq 1$ and $x_{ij} + x_{jh} - x_{ih} \leq 1$, , and the precedence relationship from i to j with a big-M: $S_j - S_i - M x_{ij} \geq p_i - M$.

The first formulation based on these variables [AVOTG93] relies on the concept of forbidden sets. A forbidden set F is a set of activities which cannot be scheduled simultaneously because the sum of their demands would exceed the cumulative capacity, so for any forbidden set F, we add a constraint to force a precedence between two activities: $\sum_{\{i,j\} \subseteq F} x_{ij} \geq 1$. There is is an exponential number of forbidden sets in general, so this formulation cannot be used in practice.

A second formulation based on the concept of resource flow was proposed in [AMR03] to obtain a more compact formulation. The principle is to indicate

the quantity of cumulative resource made available for activity j when activity i finishes. Two families of constraints are then added to the model to express that the total incoming and outgoing flow to an activity must match its demand. Two dummy activities are placed at the beginning and end of the schedule to provide and capture all the capacity. This formulation has $O(n^2)$ binary variables, $O(rn^2)$ continuous variables and $O(rn^2 + n^3)$ constraints, so it is reasonably compact. Note moreover that the model size does not depend on the scheduling horizon H.

Event-based formulations Event-based formulations, introduced in [KALM11], are based on the fact that an optimal solution to an RCPSP always exists where the start time of any activity coincides with the end time of another activity, or with the scheduling origin (a property which was already exploited implicitly by the flow formulation above), so only n events have to be scheduled. We thus have n continuous variables to represent the events dates, with an ordering constraint to break symmetry.

Two families of formulations exist, depending on how the activities are connected to the events. In the Start/End formulations, we have two binary variables for each activity/event pair: $x_{ie} = 1$ if activity i starts at event e and 0 otherwise, and $y_{ie} = 1$ if activity i ends at event e and 0 otherwise. In the On/Off formulations, we only use one binary variable for each activity/event pair: $z_{ie} = 1$ if activity i starts or is still being processed at event e.

These formulations both have $O(n^2)$ binary variables, $O(n)$ continuous variables and $O(n^3 + n(p + r))$ constraints where p is the number of precedence constraints. The linear relaxation quality is quite poor, though.

3.6 LP-based strengthening of the cumulative constraint

Many methods have been developed to calculate lower bounds on the makespan of RCPSP. As we will explain in much more detail in Chapter 4, we noticed as part of the work done in preparing this thesis that many of these methods, notably several of these relying on a linear programming formulation, as well as the new cumulative strengthening method that we present in Chapter 4, provide naturally, in addition to a lower bound, a redundant cumulative constraint. Redundant means that we do not lose a feasible solutions but obtain different demands, which can possibly be exploited to compute stronger propagations. This is another example of the integration of various algorithmic methods through Constraint Programming.

In these methods, we often first relax the RCPSP into a CuSP, that is we abandon the precedence constraints and keep only one cumulative constraint.

The first method known to us to introduce a reformulation of the cumulative constraint was based on Dual Feasible Functions. A good survey was published in [CAdC10]. DFF are functions that map the original demands to demands in a new cumulative functions, in such a way that no feasible solution is lost. In this line of research, particular classes of functions which satisfied this property, as well as a branch and bound algorithm to generate them all were exhibited. There was no reformulation giving a different demand to each activity, or taking into account special cases, such as very long activities. We are also not aware of DFF being used to strengthen propagation, but only to compute global bounds.

As for the methods based on linear programming, those computing a Linear Lower Bound, such as in [CN00], can be used to compute a reformulation as well. Indeed, studying the dual of the linear program immediately yields a redundant cumulative function.

Other methods based on linear formulations exist and could be used to compute a form of cumulative strengthening, such as [MMRB98, BK00, CN03], but those formulations are time-indexed and would yield a complex reformulation with a variable demand profile. The computations would probably be too heavy for this to be of any practical use.

Note that many other techniques to compute LB which do not yield strenghtening exist, but we do not mention them here. Information about them can be found in the survey [NAB+06], in Section 3.7 of [BK12], or in [Art16].

The Cumulative Strengthening as we introduce it in Chapter 4 is much stronger than Dual Feasible Functions while being much cheaper to compute than the formulations based on linear programming that we mentioned above.

3.7 Conflict-based search

The most effective way to do this is to look for a conflict as high up as possible in the search tree, and that for two purposes. First, the higher the conflict in the tree, the largest the volume of the search space it cuts. Second, learn a new constraint (no-good) from this conflict, in order not to direct the search anymore in this area of the search space.

For very hard feasibility problems or when trying to prove that the solution obtained is optimal (so-called exact method), we must explore the entire search space.

Some historical approaches use RCPSP-specific structures, but this is no longer the case in the latter approaches which memorize very simple decisions and operate in a much more general framework than the RCPSP.

The first such approach was introduced in [DH97] and relies on cutsets (sets of activities whose predecessors have all been scheduled already). If the search reaches a cutset which has already been processed, we can reuse the result from that earlier search which has been saved. This method was for a decade the best way of exploring the whole search space of an RCPSP and proving lower bounds.

Minimal Critical Sets were introduced in [Lab05]. They are sets of activities which violate a cumulative constraint, and thus cannot be scheduled simultaneously, so there must be a precedence between (at least) two of them. The MCS-based search thus branches on possible precedences which break MCS.

Yet another approach was introduced in [SFSW10] and consists in modeling the RCPSP as a SAT problem, using the TTEF propagation to enforce the cumulative constraint. The main idea here is to use the native clause learning mechanism of the SAT solver, using so-called TTEF explanations: when a conflict is detected by the TTEF, a clause is generated and added to the SAT formulation. The basic decisions in this formulation consists of inequalities between start dates of tasks and times. This approach works well for problems with small time horizons, but because the time is enumerated in the formulation, this approach doesn't scale to realistic time horizons.

The last approach to date is the Failure Directed Search, which we explain in more detail in Subsection 2.5.3.4.

As a general remark, these techniques (and all decision making techniques for scheduling problems, such as branching heuristics, local search techniques, and propagations) fall into two categories, depending on whether they raise a precedence between two activities or they set a temporal bound on the start or end time of an activity.

It seems that decisions on precedences have less impact than decisions on times, at least to find a solution, and that decisions on dates scale better as well. This can be seen with the extreme example of assuming that we have a perfect branching heuristic and that we will find the optimal solution by always taking the left branch at each node: with a chronological branching rule such as Set-Times, only n decisions have to be taken in this case, while n^2 decisions have to be taken if we want to raise all precedences.

If the goal is to prove optimality, the best approach is less clear. Experiments with the Failure Directed Search algorithm show that the Set-Times rules works a bit better than precedence raising, but the reason is unclear, and this observation might just come from lack of research in this direction.

Another reason to prefer taking decisions on dates is that all decision-taking methods in a constraint programming engine have to work well together, and that propagations of the cumulative constraints on precedences have not been

developed as much as propagations on dates.

As far as we know, no research has been done yet on trying to record precedence disjonctions of small cardinality as no-goods. This research direction might yield interesting results.

3.8 List scheduling

The RCPSP is often referred to as one of the most intractable problems in Operations Research, since even very small instances can not be solved to optimality with the current technology. To circumvent this limitation of exact methods, numerous RCPSP-specific heuristics have been developed. A good survey of the different techniques can be found in [HK00], but detailed experiments and comparisons reveal that all good heuristics and local search techniques boil down to list scheduling.

The general principle of list scheduling (sometimes also called Schedule Generation Schemes) consists in representing a solution in the form of a total ordering of the activities (list), which is then transformed into a schedule through a decoding procedure specific to the particular heuristic.

An example of a decoding procedure is the Earliest Start Schedule, which consists in taking the first activity in the list, scheduling it at the earliest date possible which does not conflict with an already scheduled activity, and repeating these steps until the list is empty. This particular decoding procedure has the convenient property that for an RCPSP with a regular objective function, a list of all activities always exists such that the procedure Earliest Start Schedule provides an optimal schedule. Many other decoding procedures are available in the literature.

On the other hand, this heuristic works poorly when no good decoding procedure exists for the particular constraints or objective of the problem, for example when there are many implicit precedences, or if the objective is irregular.

An efficient improvement of list scheduling procedures is called Forward-Backward Improvement. After decoding a given list, a new list is created by ranking the activities in the order of decreasing completion times. This list is decoded into a backward schedule, and the whole procedure is repeated until no new schedules are created. The best schedule which appeared in the course of the process is output.

Chapter 4

Cumulative Strengthening

As mentioned above in Section 3.6, several methods of computing lower bounds on the makespan of an RCPSP, especially those based on the energy bound, can be extended to provide a redundant cumulative constraint as well.

In this chapter, which was already published as [BB17a], we show how to construct redundant cumulative constraints.

4.1 Introduction

The objective of this work is thus not to give a new filtering algorithm, but to improve all the reasonings based on the notion of energy, which will automatically improve the existing filtering algorithms which rely on this notion, notably timetabling and edge-finding. See for example Figure 4.1, which gives a possible reformulation for the instance of Figure 3.1: the energy bound is increased from 9.75 to 11.5, and this will help the filtering algorithms adjust the time bounds for this instance.

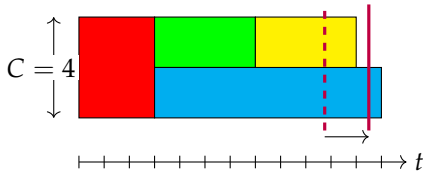


Figure 4.1: Reformulation of the instance of Figure 3.1. Energy bound: 11.5.

The reformulation of cumulative constraints that we introduce in this chapter is such that all valid schedules remain valid for the reformulated constraint, but we now have a guarantee that the energy lower bound matches the makespan of the preemptive relaxation. The *preemptive relaxation* of a cumulative resource

problem is a solution which satisfies the cumulative constraints but enables tasks to be interrupted and restarted later (compared to a non-preemptive schedule where a task of length p must run uninterrupted from its starting time t until $t + p$). These reformulations are relatively cheap to compute (after a precomputation which does not depend on the instance) and they provide a significant improvement for all algorithms which rely on energy reasonings. In our experiments, we used them within an edge-finding algorithm.

Let us start by studying two small examples for a resource of capacity $C = 3$, to get an idea of the spirit of these reformulations:

Since on such a resource a task of demand 3 cannot run in parallel with any other task, it can occupy the whole resource. Moreover, since a task of demand 2 cannot run in parallel with another task of demand 2, a valid bound on the makespan of such an instance is the sum of the lengths of tasks of demand 2 and 3. Hence, a valid reformulation for a resource of capacity $C = 3$ is to discard tasks of demand 1, and to increase the demand of tasks of original demand 2, so that they now occupy the full capacity of the resource. See Figure 4.2 for an example where this reformulation increases the energy bound from 7.33 to 8.

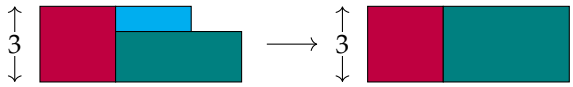


Figure 4.2: Energy bound increased from 7.33 to 8.

Again for a resource of capacity $C = 3$, we notice that the longest task of demand 1 cannot run in parallel with any task of demand 3. So a valid lower bound on the makespan is the sum of the lengths of tasks of demand 3, plus the length of the longest task of demand 1. Hence, a valid reformulation for a resource of capacity $C = 3$ is to discard tasks of demands 1 and 2, except for the longest one of demand 1, and to increase the demand of that task so that it now occupies all of the capacity of the resource. See Fig. 4.3 for an example where this reformulation increases the energy bound from 9 to 10.

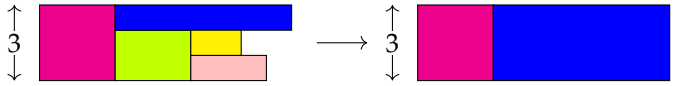


Figure 4.3: Energy bound increased from 9 to 10.

In the rest of this chapter, we will introduce a new method to compute lower bounds for the cumulative resource problem. We will show that it also yields a new, valid, cumulative constraint which can itself be passed to existing filtering algorithms for further propagation. We will then show that after a precomputation phase which is independent from the instance, computing this redundant

constraint for a given problem can be done very effectively. Then we will provide some experimental results. A similar approach was introduced in [CN07] under the name of dual feasible functions. We will show that our approach generalizes and supersedes the dual feasible functions based redundant constraints.

4.2 A compact LP for preemptive cumulative scheduling

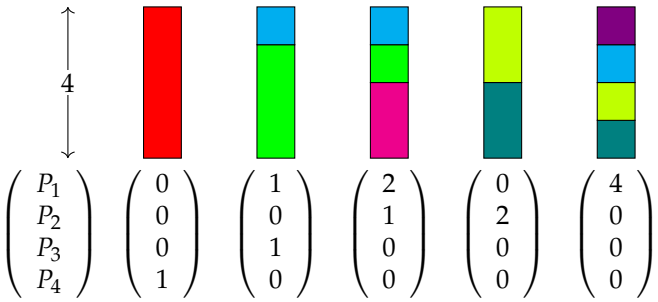


Figure 4.4: The five possible task configurations on a cumulative of capacity C=4.

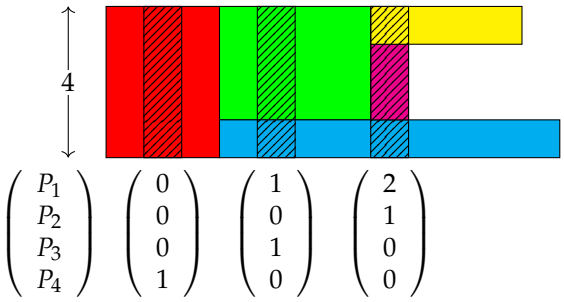


Figure 4.5: The three task configurations used to build the reformulation of Figure 3.1.

We are given a discrete cumulative resource of capacity C and a set of tasks, each with a fixed length and demand level. We want to compute a lower bound of the makespan of non-preemptive schedules of the set of tasks on the resource.

We now introduce tasks configurations: a configuration is an integer vector P of size C such that $\sum_{c=1}^{C} P_c \times c = C$. P_c is the number of tasks of demand c in configuration P. Thus, the configurations are exactly the integer partitions of

C (see Figure 4.4). We denote them $\mathcal{P}$. $\mathbf{x_P}$ can be interpreted as the time during which configuration P is run. A configuration indicates a number of tasks for each demand level which can run at the same time on the cumulative resource. See Figure 4.5 for an example, where the resource has capacity $C = 4$: the configuration $P = (2, 1, 0, 0)$ corresponds to the possibility of running 2 tasks of demand 1 and 1 task of demand 2 simultaneously on the resource. Another possible configuration could be $P = (4, 0, 0, 0)$, corresponding to running 4 tasks of demand 1 simultaneously.

Note that these are demands configurations, not tasks configurations as often seen with configuration LPs.

A key idea in our approach is the use of *non-superposition constraints*: a task cannot run in parallel of itself, so it can only use one slot on a given configuration. Generalizing this idea, any j tasks of demand c can only occupy up to j slots of demand c on a configuration, otherwise there would be at least a task occupying two slots on a configuration. This remark justifies the second set of constraints in the linear program below, called *non-superposition constraints*.

For each demand $c \leq C$, we call s_c the sum of the lengths of tasks of demand c, and for $1 \leq j \leq \lfloor \frac{C}{c} \rfloor - 1$, $s_{c,j}$ the sum of the lengths of the j longest tasks of demand c:

$$\forall 1 \leq c \leq C, s_c = \sum_{i:c_i=c} p_i$$

$$\forall 1 \leq c \leq C, \forall 1 \leq j \leq \lfloor \frac{C}{c} \rfloor - 1, s_{c,j} = \max_{\mathcal{I}:\begin{cases} |\mathcal{I}| = j \\ \forall i \in \mathcal{I}, c_i = c \end{cases}} \sum_{i \in \mathcal{I}} p_i$$

Given the set $\mathcal{P}$ of all combinations of the demands of tasks which can be run at the same time, we can show that the optimal value of the following LP is the minimal makespan for a preemptive schedule which satisfies the cumulative constraint:

$$\begin{aligned}
\min \quad & & \sum_{P \in \mathcal{P}} \mathbf{x_P} & \\
\text{s.t.} \quad & \forall c \in [1, C] & \sum_{P \in \mathcal{P}} P_c \mathbf{x_P} \geq s_c & \quad (4.1) \\
& \forall c \in [1, C], \forall j \in [1, \lfloor \tfrac{C}{c} \rfloor - 1] & \sum_{P \in \mathcal{P}} \min(P_c, j) \mathbf{x_P} \geq s_{c,j} & \quad (4.2) \\
& \forall P \in \mathcal{P} & \mathbf{x_P} \geq 0 &
\end{aligned}$$

The first set of constraints, labelled (4.1), states that for each demand level c, the total time allocated on the resource for tasks of demand c should be greater than the total length of these tasks. Indeed, P_c is the number of slots for tasks of

demand c on configuration P and $\mathbf{x_P}$ is the number of times that configuration P is selected. The sum runs over all the configurations.

The second set of constraints, labelled (4.2), are the *non-superposition constraints* as introduced above. They ensure that whenever we schedule only j tasks of the same demand level, we take into account at most j occurrences of this demand level in each configuration, so that a given task cannot occupy more than its demand in any configuration.

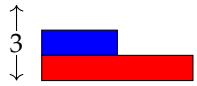


Figure 4.6: Example schedule

Let us illustrate this on a simple example: if we have a resource of capacity 3, and two tasks, both of demand 1, one of length l and the other of length $2l$, as illustrated in Figure 4.6, then without the constraints (4.2) the value of the best LP solution would be l. Indeed, with $C = 3$, the configurations are $\mathcal{P}_3 = (3,0,0),(1,1,0),(0,0,1)$. If we label these configurations respectively 0, 1 and 2, the linear program without the non-superposition constraints becomes:

$$\begin{aligned} \min \quad & \mathbf{x_0} + \mathbf{x_1} + \mathbf{x_2} \\ \text{s.t.} \quad & 3\mathbf{x_0} + \mathbf{x_1} \geq 3l \\ & \mathbf{x_0} \geq 0 \\ & \mathbf{x_1} \geq 0 \\ & \mathbf{x_2} \geq 0 \end{aligned}$$

The optimal solution to this program is $\mathbf{x_0} = l$, $\mathbf{x_1} = 0$, $\mathbf{x_2} = 0$, as shown in Figure 4.7.

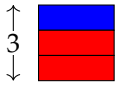


Figure 4.7: Best LP solution without the non-superposition constraints

This is a poor solution since the lower bound is not even as long as the longest task. Non-superposition constraints prevent this situation from occuring. Indeed, the non-superposition constraints in this case are:

$$\begin{aligned} \mathbf{x_0} &\geq 2l \\ 2\mathbf{x_0} &\geq 3l \end{aligned}$$

With these additional constraints, the optimal solution is now $\mathbf{x_0} = 2l$, $\mathbf{x_1} = 0$, $\mathbf{x_2} = 0$, and we get a lower bound of $2l$ on the makespan, as expected.

It is enough for the non-superposition constraints to be satisfied for any j tasks to ensure that the inequality stands with respect to the length of the j longest tasks. These constraints distinguish our reformulation from previous approaches: as we will prove now, these non-superposition constraints are quite strong: they are indeed a necessary and sufficient condition to get a preemptive schedule.

Proposition 3. *Given a discrete cumulative resource of capacity C and a set of n tasks with respective lengths p_i and demands c_i, the energy bound E of the best reformulation with non-superposition constraints is equal to the makespan M of the optimal preemptive schedule.*

Proof. We show that the energy bound E, as defined in definition 2, in our reformulation is equal to the makespan M of the optimal preemptive schedule.

First, we show that $E \leq M$. Indeed, when we have a preemptive schedule, all tasks have enough time and resources to be completed (so the constraints (4.1) in the primal are satisfied), and the non-superposition constraints are enforced (so the constraints (4.2) are satisfied). All the constraints of the primal are satisfied, and we can derive a feasible solution of the LP.

We now show that $M \leq E$. To this end, given a solution to the primal LP, that is the duration we will spend in each configuration, we will construct a preemptive schedule. Independently for each demand level $c \in [1, C]$, we construct a *staircase*, representing the slots available to tasks of demand c. The staircase for tasks of demand c is built in the following way: we consider the configurations $P \in \mathcal{P}$ in the order of decreasing values of P_c, and for each of them we add to the staircase a column of demand P_c and width $\mathbf{x_P}$. We can interpret the staircase as a superposition of $\left\lfloor \frac{C}{c} \right\rfloor$ horizontal lines, ordered by lengths (the shortest lines on the top, the longest at the bottom). We denote l_1, l_2, ..., $l_{\lfloor \frac{C}{c} \rfloor}$ the lengths of the lines, with $l_1 \geq l_2 \geq ... \geq l_{\lfloor \frac{C}{c} \rfloor}$. See Figure 4.8 for an example showing different configurations with slots of demand c, shown in white, and slots of other demand levels, shown in dashed lines. On this example, we have a cumulative of capacity 9, and 5 tasks of demand 2 and of respective lengths 15, 13, 9, 8 and 6.

The lines can be interpreted as independent machines, and we now assign the tasks of demand c to the configurations in the following manner, similar to McNaughton's method [McN59]: we start with the longest task, and place it on the topmost line, starting from the rightmost free slot. If we fill the whole line, we continue on the next line, starting again from the right, without going further than the rightmost slot occupied by the task on the line above, so that a task does not occupy two slots of the same configuration. If necessary, we repeat

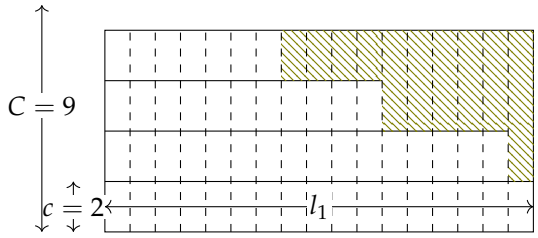


Figure 4.8: Example of configurations forming a staircase of slots of demand c.

until we can place all the task on the staircase. See Figure 4.9 for an example of placing one task. The goal is to preserve a staircase structure after removing the positions covered by the longest task. We then repeat this process with the following tasks, considered in the order of decreasing lengths. See Figure 4.10 for an example, the positions on each line are marked with the number of a task.

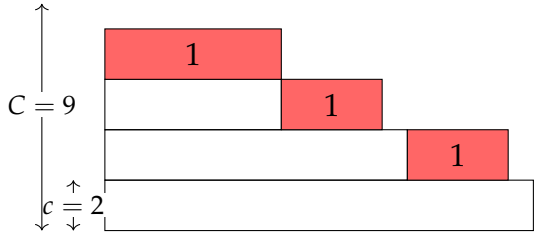


Figure 4.9: Placing one task on the staircase.

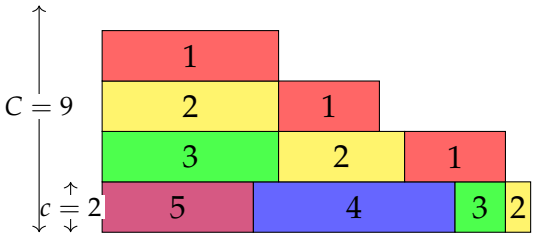


Figure 4.10: Preemptive schedule obtained from the staircase.

Let us show that a schedule constructed in this fashion is preemptive. There is enough time for all of the tasks to run (constraints (4.1) of the primal), so we will never have to exceed the capacity of the last line. What remains to prove is that no job has to be scheduled twice on the same configuration. We will prove this by showing that the staircase invariant $\forall k \leq \left\lfloor \frac{C}{c} \right\rfloor, \sum_{j=1}^{k} l_j \geq \sum_{j=1}^{k} p_j$ (where $p_1 \geq p_2 \geq ... \geq p_k$) is preserved after placing the longest task as described above.

The case $k = 1$ is obvious: there is only one line, so there can not be a superposition. Otherwise, assume that the invariant is satisfied for the staircase of lengths l. We will place the longest task according to the rule above and show that the invariant is still satisfied. We define s as the largest integer such that $p_1 \in [l_{s+1}, l_s[$. All the lines of index greater than s will be completely filled when placing the longest task. We call l' the length of the staircase lines after placing the task. There are three cases to distinguish:

If $k < s$, then $\sum_{j=1}^{k} l'_j = \sum_{j=1}^{k} l_j \geq \sum_{j=1}^{k} p_j \geq \sum_{j=1}^{k} p_j - p_1$. (The task p_1 is not placed on line k).

If $k = s$, then $\sum_{j=1}^{k} l'_j = \sum_{j=1}^{k} l_j - (p_1 - l_{s+1}) = \sum_{j=1}^{k+1} l_j - p_1 \geq \sum_{j=1}^{k+1} p_j - p_1 \geq \sum_{j=1}^{k} p_j - p_1$. (The task p_1 occupies the end of line k).

If $k > s$, then $\sum_{j=1}^{k} l'_j = \sum_{j=1}^{k} l_j - (p_1 - l_{s+1}) - \sum_{j=s+1}^{k} (l_j - l_{j+1}) = \sum_{j=1}^{k+1} l_j - p_1 \geq \sum_{j=1}^{k+1} p_j - p_1 \geq \sum_{j=1}^{k} p_j - p_1$. (The task p_1 occupies all of line k).

This proves that by construction, one can always place the longest task on the staircase with no superposition, the remaining spots are still staircase-shaped, and the total staircase surface is equal to the total task surface.

Using this construction for each demand level, we can assign locations on the configurations to tasks independently for the tasks of a given demand. We repeat this construction for every demand $c \in [1, C]$. Since we do not consider time or precedence constraints, changing the order in which configurations are run preserves a preemptive schedule. □

4.3 Reformulation

We now consider the dual of the previous linear programming formulation:

$$\begin{aligned}
\max \quad & \sum_{c=1}^{C} \left(s_c \mathbf{h_c} + \sum_{j=1}^{\lfloor \frac{C}{c} \rfloor - 1} s_{c,j} \mathbf{h_{c,j}} \right) \\
\text{s.t.} \quad \forall P \in \mathcal{P} \quad & \sum_{c=1}^{C} \left(P_c \mathbf{h_c} + \sum_{j=1}^{\lfloor \frac{C}{c} \rfloor - 1} \min(P_c, j) \mathbf{h_{c,j}} \right) \leq 1 \\
& \forall c \in [1, C] \quad (\mathbf{h_c} \geq 0 \text{ and } \forall j \in [1, \lfloor \frac{C}{c} \rfloor - 1] \; \mathbf{h_{c,j}} \geq 0)
\end{aligned}$$

Here we introduce variables $\mathbf{h_c}$ for each $c \in [1, C]$ and $\mathbf{h_{c,j}}$ for each $c \in [1, C]$ and $1 \leq j \leq \lfloor \frac{C}{c} \rfloor - 1$, corresponding to constraints in the primal linear program. The constraints state that all reformulations must remain feasible, and we search for the reformulation which gives the largest energy reasoning bound.

This dual formulation is particularly interesting in that we can interpret the dual variables as corresponding to demands of a new valid cumulative constraint, which can be used with all filtering techniques for cumulative constraints, notably edge-finding. In contrast, previous linear programming approaches to this problem (e.g. [BK00, BD04]) used formulations similar to the primal formulation above, but did not make use of the new information about the cumulative constraint.

More formally, we can define a new valid cumulative constraint as follows:

Proposition 4. *Given a solution to this linear program, if a cumulative resource of capacity C is reformulated with a capacity of* 1*, and that the tasks are ajusted so that the demand of the i^{th} longest task of initial demand c becomes $h_c + \sum_{j \geq i} h_{c,j}$, then all previously feasible schedules are still feasible with the new cumulative resource.*

Proof. Schedules which are feasible with the original cumulative resource are such that at any point in time, the tasks I being executed satisfy the constraints $\sum_{i \in I} c_i \leq C$, so the set of demands of the tasks being executed is contained in one of the configurations P of $\mathcal{P}$.

We can now rewrite the constraint of the linear program above as follows:

$$
\begin{aligned}
&\sum_{c=1}^{C} \left(P_c \mathbf{h_c} + \sum_{j=1}^{\lfloor \frac{C}{c} \rfloor - 1} \min(P_c, j) \mathbf{h_{c,j}} \right) \leq 1 \\
\Leftrightarrow &\sum_{c=1}^{C} \left(\sum_{i=1}^{P_c} h_c + \sum_{j=1}^{\lfloor \frac{C}{c} \rfloor - 1} \sum_{i=1}^{\min(P_c, j)} \mathbf{h_{c,j}} \right) \leq 1 \\
\Leftrightarrow &\sum_{c=1}^{C} \left(\sum_{i=1}^{P_c} h_c + \sum_{i=1}^{P_c} \sum_{j=i}^{\lfloor \frac{C}{c} \rfloor - 1} \mathbf{h_{c,j}} \right) \leq 1 \\
\Leftrightarrow &\sum_{c=1}^{C} \sum_{i=1}^{P_c} \left(h_c + \sum_{j=i}^{\lfloor \frac{C}{c} \rfloor - 1} \mathbf{h_{c,j}} \right) \leq 1
\end{aligned}
$$

This constraint precisely states that for any configuration $P \in \mathcal{P}$, if we change the demands of all the tasks in such a way that the demand of the i^{th} longest task of initial demand c becomes $h_c + \sum_{j \geq i} h_{c,j}$, then the new demands of the tasks in P will sum to less than 1. This proves that the schedule is still feasible at any point in time after the reformulation. □

In constraint-based scheduling, the demands are traditionally integer numbers, while the demands given in the proposition above are fractional and smaller

than 1. In an implementation, one can multiply each value by the LCM of the denominators of the new demands, to work with integer values only.

This reformulation is stronger than the original constraint, and because the energy lower bound is equal to the minimal makespan for a preemptive schedule, as we have shown previously, the standard propagation algorithms for cumulative constraints should propagate better with this constraint than with the original constraint. Before we can do so, we have to find a way to compute it in an efficient way.

4.4 Precomputing the vertices

Note that since this reformulation is a mapping of the task capacities, the size of the underlying polytope does not depend on the number of tasks involved, but only on the capacity of the resource. Note also that the instance-specific data, the s_c and $s_{c,j}$ values, only appear in the objective function but not in the constraints. Thus we can precompute the vertices of the polytope associated with this linear program. A convenient way of doing it is to enumerate the vertices of this polytope using *LRS* [Avi00]. To solve this program we then just have to evaluate its objective function over the vertices which were precomputed.

Using a dominance property we show that on some of the vertices, none of the objective functions we consider will reach its maximum, and we eliminate these vertices. Indeed, we are only interested in those vertices that lead to reformulations in which at least one task gets assigned a higher demand than in another formulation. In other words, a reformulation h (solution vector to the LP above) is dominated if there exists a reformulation h', such that $\forall i, \forall c, h_c + \sum_{j \geq i} h_{c,j} \leq h'_c + \sum_{j \geq i} h'_{c,j}$. We eliminate dominated reformulations. Reciprocally, we noticed in our experiments that all remaining reformulations are useful and can yield the best bound on different instances.

After eliminating all the dominated vertices, the number of vertices that remain is relatively small, and is practical up to $C = 12$ (from about hundred thousands vertices in the polytope, only a few hundreds vertices are not dominated).

C	1	2	3	4	5	6	7	8	9	10	11	12
number of vertices	1	2	4	7	12	22	38	67	124	222	392	734

4.5 Discussion

As an example, here are the non-dominated solutions for C=3:

$h_{1,1}$	$h_{1,2}$	h_1	h_2	h_3
0	0	$\frac{1}{3}$	$\frac{2}{3}$	1
0	0	0	1	1
0	$\frac{1}{2}$	0	$\frac{1}{2}$	1
1	0	0	0	1

The first row corresponds to no reformulation. The second row gives a bound which is the sum of the lengths of tasks of demand 2 and 3 (since no two tasks of demand 2 can be executed at the same time), and was used to reformulate the instance in Figure 4.2. The third row is the most interesting, and gives a bound which is the length of tasks of demand 3, plus half the length of tasks of demand 2, plus half the length of the two longest tasks of demand 1. Finally, the fourth row gives a bound which is the length of tasks of demand 3, plus the length of the longest task of demand 1, and was used to reformulate the instance in Figure 4.3.

It is interesting to notice that if all tasks are of demand 1, we find McNaughton [McN59] bound for parallel machines. Indeed, in this case the linear program reduces to:

$$\begin{aligned} \max \quad & s_1\mathbf{h_1} + \sum_{1\leq j<C} s_{1,j}\mathbf{h_{1,j}} \\ \text{s.t.} \quad & C\mathbf{h_1} + \sum_{1\leq j\leq C-1} j\mathbf{h_{1,j}} \leq 1 \\ & (\mathbf{h_1} \geq 0 \text{ and } \forall j \in [1, C-1]\ \mathbf{h_{1,j}} \geq 0) \end{aligned}$$

The optimal solution to this linear program is $\max(\frac{s_1}{C}, s_{1,1})$, which is McNaughton's bound.

4.6 Comparison with dual feasible functions

Dual feasible functions, used in [CN07] to get lower bounds for the energy of a set of tasks on a cumulative resource, can also be used to reformulate cumulative resources in a similar fashion to what was done above, by looking at the dual formulation.

The valid reformulations obtained using dual feasible functions change the demand of all tasks of original demand c into h_c, for h_c values which satisfy the

following program:

$$\begin{array}{lll} \max & & \sum_{c=1}^{C} s_c \mathbf{h_c} \\ \text{s.t.} & \forall P \in \mathcal{P} & \sum_{c=1}^{C} P_c \mathbf{h_c} \leq 1 \\ & \forall c \in [1, C] & \mathbf{h_c} \geq 0 \end{array} \tag{4.3}$$

We can notice that the constraint polytope of this program is the master knapsack polytope.

In a similar fashion, the valid reformulations that we introduce change the demand of the i^{th} longest task of initial demand c into $h_c + \sum_{j \geq i} h_{c,j}$, for h_c and $h_{c,j}$ values which satisfy the following program:

$$\begin{array}{lll} \max & & \sum_{c=1}^{C} s_c h_c + \sum_{c=1}^{C} \sum_{1 \leq j < C/c} s_{c,j} h_{c,j} \\ \text{s.t.} & \forall P \in \mathcal{P} & \sum_{c=1}^{C} P_c h_c + \sum_{c=1}^{C} \sum_{j < C/c} \min(P_c, j) h_{c,j} \leq 1 \\ & \forall c \in [1, C] & (h_c \geq 0 \text{ and } \forall j \in [1, C/c[\ h_{c,j} \geq 0) \end{array} \tag{4.4}$$

Proposition 5. *The reformulations induced by dual feasible functions are dominated by the preemptive reformulations.*

Proof. All solutions of (4.3) are feasible for (4.4), if we set all the $\mathbf{h_{c,j}}$ variables to 0, with the same objective value, so the optimal preemptive reformulation dominates the optimal DFF reformulation. □

4.7 Additional constraints

We can stighly strenghten this formulation by adding, in the primal program, constraints which state that $\forall c \in [1, C[, \sum_{P \in \mathcal{P}} \min(P_c, 1) \mathbf{x_P}$ must be greater than the minimum length of a bin in a bin-packing of the tasks of demand c into $\frac{C}{c}$ bins.

We call these additional constraints *bin-packing constraints*. They are justified by the fact that each configuration contains at most $\frac{C}{c}$ slots for tasks of demand c, and that the assignment of tasks to slots can be viewed as a bin-packing problem.

One possibility to add these contraints is to compute an optimal bin-packing of the tasks of demand c for each $c \in [1, C[$.

Another possibility is to use the bound of Marcello and Toth [LMM02], which in this case reduces to $\forall c \in [1, C[,\ \sum_{P \in \mathcal{P}} \min(P_c, 1)\mathbf{x_P} \geq p_{\frac{C}{c}} + p_{\frac{C}{c}+1}$.

These two ways of formulating the constraint are both convenient once translated into the dual formulation, which we use to compute the redundant cumulative resource.

4.8 Experiments and results

We ran some experiments by manually adding redundant constraints to RCPSP instances from the PSPLIB [KS97]. In practice, it is too expensive to try all of the reformulations. We have to rely on heuristics to select them, hoping that it will strengthen the propagations enough. The best results were obtained when adding a single redundant constraint per cumulative constraint in the original problem. Choosing the redundant cumulative constraint with the highest global lower bound (highest objective value in the linear program) worked the best also to improve the propagations made by edge-finding.

A completely different problem, which we do not address in this Chapter, consists in choosing the best reformulation. We made some progress in this direction, which we report in the Outlook chapter of this thesis, in Section 6.2.

We solved the instances with CP Optimizer 12.3 running on a Core i5-2520m processor with a time limit of 10 minutes. We used the default search strategy (Restart), and we set the `IloCP::CumulFunctionInferenceLevel` parameter to `IloCP::Medium`, which activates the edge-finding algorithm described in [Vil11].

Our results show that on average, the energy lower bounds we compute are about 10% better than with the original cumulative resource. This compares to an approximate 6% improvement when using a DFF over the original bounds, but the advantage of our reformulations over DFF is stronger when we consider sets with a small number of tasks, such as the sets on which the edge-finder algorithm typically propagates. These stronger bounds enable stronger propagations, and by using destructive bounds we managed to improve the lower bounds for some of the J60 RCPSP instances of the PSPLIB:

inst	LB	inst	LB	inst	LB	inst	LB	inst	LB
9_1	85	9_5	82	9_6	107	9_7	105	9_10	90
13_1	106	13_3	84	13_4	99	13_7	82	13_8	115
13_9	96	13_10	113	25_2	96	29_1	98	29_6	144
29_7	115	41_3	90	41_10	106	45_1	90		

4.9 Conclusion

It is remarkable that our LP formulation for computing the makespan of a preemptive schedule on a cumulative resource is independent of the number of tasks, and depends only on the capacity of the resource. This formulation gives a new light on the structure of preemptive relaxations to cumulative problems.

Since all parameters specific to the instance can be encoded in the objective function only, this enables us to precompute the vertices of the LP polytope, so that solving the LP in practice is extremely fast.

From a constraint programming point of view, this formulation yields redundant cumulative resources which can be exploited by other algorithms for cumulative resources such as edge-finding.

Note also that our reformulation subsumes the Dual Feasible Functions reformulation [CN07].

Using a very similar method to the one shown in this chapter, other results from the literature, such as [MMRB98, BK00, CN03], could also be turned into a redundant cumulative constraint, but the computations would be much more expensive. We think that our approach of aggregating the activities by similar demands, except for the few longest ones, hits an interesting trade-off between the quality of the bound (notably since we can prove that it is equivalent to a preemptive relaxation of the problem) and the size of the formulation, which stays independent from the number of tasks.

Chapter 5

Fast Energy Reasoning

In this chapter, we show how to reduce the complexity of the Energy Reasoning propagation, introduced above in Subsection 3.4.5, from $O(n^3)$ to $O(n^2 \log n)$. This chapter has been published in preliminary form as [Bon16].

5.1 Introduction

Energy reasoning has been known for 25 years and is the strongest propagation known for the cumulative constraint (with the exception of not-first, not-last, none of those propagations dominating each other) for the cumulative constraint (full propagation is NP-hard). Improving the propagation of energy reasoning has sparked some interest recently [DP14], but the best algorithm to date has a complexity of $O(n^3)$ for n tasks. This high complexity is the reason why this propagation is seldom used in practice and other, weaker but faster, propagations were introduced.

In this chapter, we introduce techniques to propagate energy reasoning in time $O(n^2 \log n)$. This is an important theoretical advance to a long-standing open question. Moreover, our experiments suggest that this algorithm should also be of practical interest for difficult problems.

Our approach is based on three novel ideas, which form the next sections of the chapter. First, we proved that the so-called *additional rule* supersedes the other energy reasoning rules in the literature, and we can study this case only. We then show that detecting a propagation with the additional rule reduces to computing the maximum of a set of piecewise affine functions with special properties that we make use of. Finally, we give an algorithm to efficiently compute this maximum by using the point-line duality of projective geometry to reduce the problem to a convex hull computation. We conclude by discussing this new method and giving experimental results.

5.2 Energy reasoning rules

Given a cumulative resource of capacity C, tasks of consumption c_i and of length p_i, we respectively denote est_i and lst_i their earliest starting time and latest starting time, in the context of constraint-based scheduling. We say that a task is *left-shifted* if we try to schedule it as early as possible, that is from est_i to $est_i + p_i$, and similarly we say that it is *right-shifted* if we try to schedule it as late as possible, that is from lst_i to $lst_i + p_i$.

We also denote $W_i(a, b)$ with $b \geq a$ the *intersection energy* of task i in the interval $[a, b)$, that is $W_i(a, b) = c_i \cdot \min(b - a, p_i^+(a), p_i^-(b))$ with $p_i^+(a)$ being the length of time during which task i executes after a if it is left-shifted, and $p_i^-(b)$ being the length of time during which task i executes before b if it is right-shifted. The expression of $p_i^+(a)$ in terms of the est_i and lst_i variables is $p_i^+(a) = \max(0, \min(p_i, est_i + p_i - a))$ and $p_i^-(b) = \max(0, \min(p_i, b - lst_i))$.

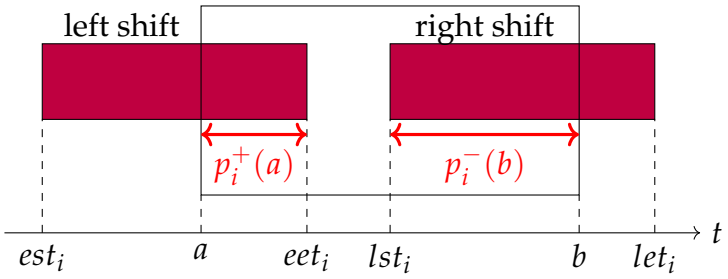


Figure 5.1: Intersection length

$W_{\neq i}(a, b)$ is the sum of the intersection energies of all tasks different from i: $W_{\neq i}(a, b) = \sum_{j \neq i} W_j(a, b)$.

Finally, $W(a, b)$ is the sum of the intersection energies of all tasks: $W(a, b) = \sum_i W_i(a, b)$.

In the rest of this chapter, the propagations will be described in the case when we shift the task under consideration as much as possible to the left. There is obviously a symmetric version of all of them in the case when we shift the task to the right.

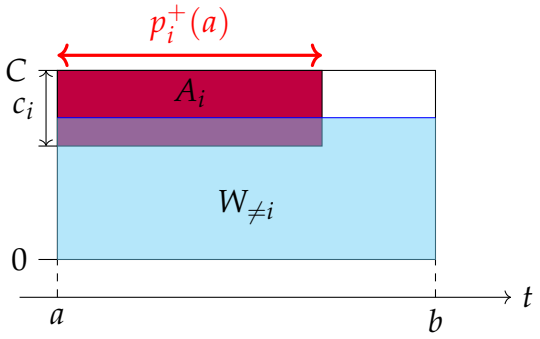
The energy reasoning rule is:

$$\text{If } W_{\neq i}(a, b) + c_i \cdot \min(b - a, p_i^+(a)) > C(b - a)$$

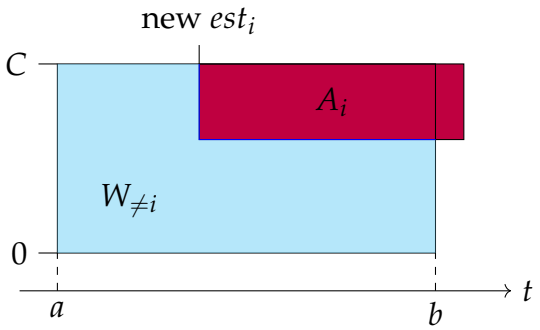
$$\text{then } est_i \geq b - \frac{C(b - a) - W_{\neq i}(a, b)}{c_i}$$

.

One can show that this rule supersedes the other rules in the literature.



(a) Energy overflow in $[a, b[$



(b) After applying Energy Reasoning

Figure 5.2: Energy Reasoning

5.3 Propagation conditions

We will now establish a simple condition to detect when the energy reasoning rule applies. Its application condition is:

$$\begin{aligned} & C \cdot (b-a) - c_i \cdot \min(p_i^+(a), b-a) < W_{\neq i}(a,b) \\ \Leftrightarrow\ & C \cdot (b-a) - c_i \cdot \min(p_i^+(a), b-a) < W(a,b) - W_i(a,b) \\ \Leftrightarrow\ & C \cdot (b-a) - W(a,b) < c_i \cdot \min(p_i^+(a), b-a) - W_i(a,b) \end{aligned}$$

The left-hand side of this inequality is constant on a time window, and the right-hand side depends on task i only. For each window, we thus compute $\max_i \left(c_i \cdot \min(p_i^+(a), b-a) - W_i(a,b)\right)$. If this value does not exceed $C \cdot (b-a) - W(a,b)$, there is no excess of intersection energy. If the maximum exceeds $C \cdot (b-a) - W(a,b)$, we know that we should propagate on the task $\operatorname{argmax}_i \left(c_i \cdot \min(p_i^+(a), b-a) - W_i(a,b)\right)$ (and maybe on other tasks, but since our goal is just to detect if there is or not a propagation, we ignore these for now).

5.4 Efficient detection of intervals with an excess of intersection energy

As we saw in the previous section, we can find all intervals on which to propagate if we compute $\text{argmax}_i \left(c_i \cdot \min(p_i^+(a), b-a) - W_i(a,b)\right)$ for all windows $[a,b)$. We will now show a way of finding the maximum of $c_i \cdot \min(p_i^+(a), b-a) - W_i(a,b)$ for all dates b in time $O(n \log n)$.

We define the function $f_{i,a}(b) := c_i \cdot \min(p_i^+(a), b-a) - W_i(a,b)$ for $b \geq a$. This function is continuous and has a very simple shape in relation with the positions of the left and right shifts, as shown on Figure 5.3.

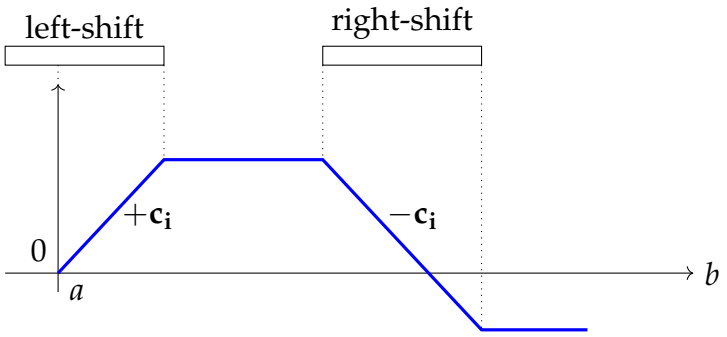
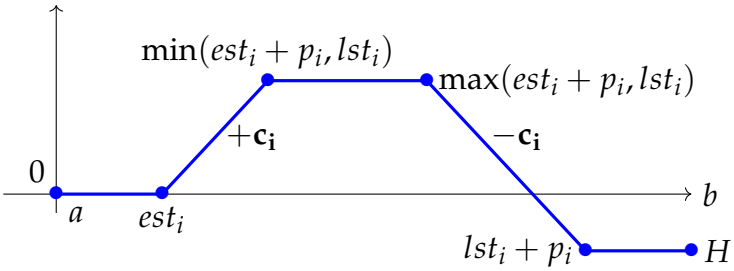


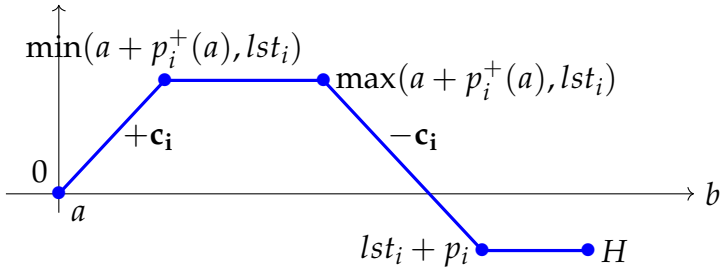
Figure 5.3: General shape of $f_{i,a}$.

More precisely, there are three cases to distinguish, depending on the relative positions of a and the two shifts. In all of these three cases, the function is continuous, piecewise-linear, with a slope of $+c_i$, 0 or $-c_i$ depending on the piece, and the coordinates of the slope changes are indicated on Figure 5.4. H is the planning horizon (upper bound on the makespan). There are three cases to distinguish, depending on the relative position of a with est_i and lst_i:

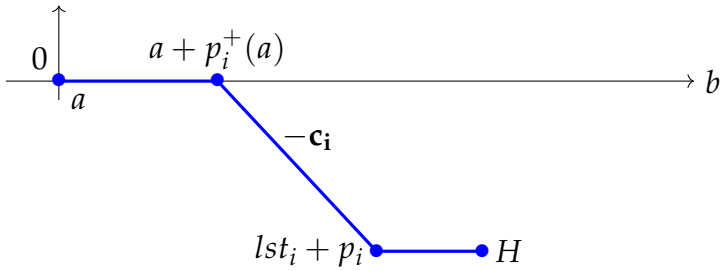
- When $est_i > a$, $f_{i,a}$ is constant and equal to 0 from a to est_i, has a slope $+c_i$ to $\min(est_i + p_i, lst_i)$, is constant to $\max(est_i + p_i, lst_i)$, has a slope $-c_i$ to $lst_i + p_i$, and is constant to H.
- When $est_i \leq a < lst_i$, $f_{i,a}$ has a slope $+c_i$ to $\min(a + p_i^+(a), lst_i)$, is constant to $\max(a + p_i^+(a), lst_i)$, has a slope $-c_i$ to $lst_i + p_i$, and is constant to H.
- Finally, when $a \geq lst_i$, $f_{i,a}$ is constant to $a + p_i^+(a)$, has a slope $-c_i$ to $lst_i + p_i$, and is constant to H.



(a) $f_{i,a}$ when $est_i > a$.



(b) $f_{i,a}$ when $est_i \leq a < lst_i$.



(c) $f_{i,a}$ when $a \geq lst_i$.

Figure 5.4: Shape of $f_{i,a}$ in different cases.

Remember that we want to compute the supremum over i of the $f_{i,a}$ functions. Since these functions have a very special structure, we will resort to geometry instead of analysis to compute their supremum. Specifically, we decompose these functions into line segments, and use the algorithm described in section 15.3.2 of [BY98] to compute the upper envelope of these segments. This will exactly yield the function $\max_i f_{i,a}$.

In our context, this algorithm works as follows: the end dates of the $O(n)$ line segments are projected on the x-axis and define $O(n)$ non-overlapping intervals. A balanced binary tree is built, whose leaves are these non-overlapping intervals in ascending order. To a node of the tree corresponds the union of the intervals of the leaves of the subtree. Each line segment is assigned to the node

deepest in the tree which contains the projected endpoints of the segment. The upper envelope of all the segments assigned to one node can be computed in time linear with the number of segments at the node. Now, since the upper envelopes of two nodes at the same level in the tree do not overlap (by construction of the tree), we can merge them in linear time, and we can merge all the envelope of one level of the tree in time $O(n \log n)$. Finally, we can merge the envelopes of the $O(\log n)$ levels of the tree in time $O(n\alpha(n) \log \log n)$ (cf. section 15.3.2 of [BY98]), which is $O(n \log n)$.

We now have an algorithm to compute the supremum of $O(n)$ line segments in time $O(n \log n)$.

5.5 Complete algorithm and complexity analysis

An important property of energy reasoning, which we make use of, is that when we have n tasks, only $O(n)$ starting dates a have to be considered as candidates for the intervals $[a, b)$ [EL90, BPN01, DP14]. More precisely, when the tasks are left-shifted, the a dates to be considered are the $O(n)$ dates in the set $O_1 = \{est_i, 1 \leq i \leq n\} \cup \{lst_i, 1 \leq i \leq n\} \cup \{est_i + p_i, 1 \leq i \leq n\}$ (cf. Proposition 19 in chapter 3 of [BPN01]). With our algorithm, we will study at once all dates b located after a. Actually, even less dates a can be studied by using the stronger characterization from [DP14].

Therefore there are only $O(n)$ interesting starting dates to consider, on which we might propagate. We start by precomputing all $O(n^2)$ values $W(a, b)$ in time $O(n^2)$ with the algorithm described in section 3.3.6.2 of [BPN01].

Then, for every interesting date a, we use the algorithm of the previous section to detect interesting intervals. Since there are $O(n)$ dates to study and that the study of each of them has complexity $O(n \log n)$, the complexity of this step is $O(n^2 \log n)$. If we find a propagation, we adjust the earliest starting time of the task according to the additional energy reasoning rule, which is done in constant time. Finally, the total complexity with this algorithm of finding one date adjustment, or finding that there is nothing to propagate, is $O(n^2 \log n)$.

Moreover, we run the energetic checker (cf. [BPN01]) before our algorithm. The energetic checker is a $O(n^2)$ test that either guarantees that energy reasoning will not find a propagation, in which case we do not need to run the energy reasoning, or that it will find one without telling which one, in which case we do run the energy reasoning.

Input: n: number of tasks
$p_1, \ldots, p_n$: length of the tasks
$c_1, \ldots, c_n$: consumption of the tasks
$est_1, \ldots, est_n$: earliest start dates of the tasks
$lst_1, \ldots, lst_n$: latest start dates of the tasks
H: planning horizon
C: cumulative resource capacity
Side Effect: Update a value est_i if Energy Reasoning propagates
if the energetic checker finds no possible propagation **then**
  return
end
Compute all $W(a, b)$ values using the algorithm of section 3.3.6.2 of [BPN01].
for $a \in \{est_i, 1 \leq i \leq n\} \cup \{lst_i, 1 \leq i \leq n\} \cup \{est_i + p_i, 1 \leq i \leq n\}$ **do**
  for $i \leftarrow 1$ ***to*** n **do**
    $I \leftarrow \emptyset$
    if $est_i > a$ **then**
      Add to I the 5 segments of figure 2, label them with i
    end
    else
      if $est_i \leq a < lst_i$ **then**
        Add to I the 4 segments of figure 3, label them with i
      end
      else
        Add to I the 3 segments of figure 4, label them with i
      end
    end
  end
  Compute the upper envelope of the segments in I using the algorithm of section 15.3.2 of [BY98]. Preserve the segments labels. Call this function f_a.
  for $b \in$ endpoints of the segments of f_a **do**
    if $C \cdot (b - a) - W(a, b) < f_a(b)$ **then**
      $i \leftarrow$ label of current segment
      $est_i \leftarrow b - \frac{C \cdot (b-a) - W(a,b) + W_i(a,b)}{c_i}$
    end
  end
end

Algorithm 1: Fast Energy Reasoning

5.6 Discussion

In a similar fashion to what is done with classical algorithms to propagate the energy reasoning, we can then re-apply this algorithm on the adjusted dates until we reach a fixpoint and there is nothing else to propagate.

Since this algorithm will detect at least one excess of intersection energy if there is one, and we showed already that energy reasoning propagates exactly when there is an excess of intersection energy, this algorithm will make at least one energy reasoning adjustment if the classical $O(n^3)$ algorithm would have done so. The propagation we get is equivalent to the $O(n^3)$ algorithm.

One difference though is in the rare case where the intersection energy is exceeded for several tasks on the same interval $[a, b)$. In this case this algorithm will only tighten the bound for the task with the highest excess of interval energy, while the original algorithm would have made the adjustments for all the tasks. We believe that in the context of constraint programming, when propagations are rare and we need to run the algorithm iteratively until we reach the fix point anyway, the relevant question is "Given that the $O(n^2)$ checker reports that there is something to propagate, how to find one such propagation as quickly as possible?" In this case our algorithm needs time $O(n^2 \log n)$ versus $O(n^3)$ for the original algorithm. More generally, if there are $k \leq n$ tasks for which to propagate, our algorithm needs time $O(kn^2 \log n)$ versus $O(n^3)$ for the original algorithm and thus constitutes an improvement.

Indeed, in a complete CP solver, global constraints are called from a propagation queue, and the heaviest propagations are only used when lighter propagations cannot reduce the domains anymore. In practice, once the Energy Reasoning algorithm finds a propagation, it will be stopped there and all lighter propagations will be tried again on the newly reduced instance for further (and less costly) improvements.

We implemented this algorithm on top of IBM Ilog CP Optimizer 12.6 and noticed that this improved algorithm gives a performance improvement in practice, on hard instances which require the use of energy reasoning.

Chapter 6

Outlook

We have presented in the previous chapters two new techniques for dealing with the cumulative constraint. These offer a theoretical contribution by the use of new methods in constraint programming, and they both have a practical interest in solving difficult feasibility problems. This is particularly the case with Failure Directed Search, where the combined use of Cumulative Strengthening and fast Energy Reasoning offer the potential to solve previously inaccessible feasibility problems. More generally, these improvements lay the path for further improvements in cumulative scheduling, of which we give a selection in this section.

6.1 Static selection of a reformulation

In practice, it is far too costly and counterproductive to use all the cumulative reformulations that can be generated by Cumulative Strengthening. Thus, there is still a lot of work to be done in determining which reformulations will or will not help a given propagation algorithm in a given context. Research tracks were laid down during a visit to François Clautiaux in Bordeaux.

Indeed, it seems that the gap to the optimal reformulation reduces as an inverse exponential of the number of reformulations considered. It is therefore probable that a small number of reformulations will make it possible to obtain the bulk of the power of this technique at a very low cost. We have two directions to study this phenomenon: one is experimental, by measuring the contribution of a small number of randomly chosen reformulations, and the other is theoretical, aiming at quantifying the energy improvement offered by a given set of reformulations. We have already identified some properties to characterize the cumulative resources for which a given reformulation will improve the energy bound.

Finally, another direction is to try to adapt the work on Vector Packing Dual Feasible Functions (VP-DFF) to quickly compute reformulations which take into account several cumulative constraints on the same intervals.

6.2 Dynamic computation of a reformulation within propagators

A second line of research consists in computing reformulations directly within the propagation algorithms, in order to exploit the full potential of cumulative strengthening. In this context, it is also possible to exploit more information than the cumulative constraints only, as described above. Notably, the state of the timetable as well as the precedence graph can be used. This allows for much stronger reformulations. I directed the M1 internship of Réda Bousseta on this subject and I gave a preliminary presentation on this work at the ROADEF 2017 conference [BB17b].

We show in this work how to compute, within the Edge-Finder, reformulations adapted to the set of activities being examined. This makes it possible to obtain much stronger bounds over this set than with a reformulation of the whole instance. The basic Edge-Finder, thus enhanced, gains considerable propagation strength, and we show that it allows for deductions that the unreinforced Timetable Edge-Finder can not find.

We also show how to integrate the same idea within Precedence Energy and Energy Reasoning.

6.3 Integration with the Failure Directed Search

Finally, the newly introduced Failure Directed Search technique (see Subsection 2.5.3.4) to improve the dual bounds for very difficult instances, such as those from the PSPLIB, are based on using strong propagations. The two techniques introduced in this thesis contribute to this: the $O(n^2 \log n)$ Energy Reasoning can be used instead of the weaker algorithms previously used, and the Cumulative Strengthening can reinforce the deductions made by Energy Reasoning.

Using these two tools jointly, within the Failure Directed Search, offers the potential to dramatically improve dual bounds for certain open instances of the PSPLIB which have been long-standing and hitherto unattainable.

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, April 1993.

[ABED+15] Emmanuel Agullo, Olivier Beaumont, Lionel Eyraud-Dubois, Julien Herrmann, Suraj Kumar, Loris Marchal, and Samuel Thibault. Bridging the Gap between Performance and Bounds of Cholesky Factorization on Heterogeneous Platforms. pages 34–45. IEEE, May 2015.

[ADN13] Christian Artigues, Sophie Demassey, and Emmanuel Néron. *Resource-Constrained Project Scheduling: Models, Algorithms, Extensions and Applications*. John Wiley & Sons, March 2013.

[AMR03] Christian Artigues, Philippe Michelon, and Stéphane Reusser. Insertion techniques for static and dynamic resource-constrained project scheduling. *European Journal of Operational Research*, 149(2):249–267, September 2003.

[Art16] C. Artigues. Méthodes exactes pour l'ordonnancement de projets à moyens limités (RCPSP). In *17eme congrès annuel de la société Française de Recherche Opérationnelle et d'Aide à la Décision (ROADEF 2016)*, Compiègne, France, 2016. Invited plenary talk.

[Avi00] David Avis. LRS: A Revised Implementation of the Reverse Search Vertex Enumeration Algorithm. In *Polytopes — Combinatorics and Computation*, pages 177–198. Birkhäuser Basel, 2000.

[AVOTG93] Ramón Alvarez-Valdés Olaguíbel and José Manuel Tamarit Goerlich. The project scheduling polyhedron: Dimension, facets and lifting theorems. *European Journal of Operational Research*, 67(2):204–220, June 1993.

[BB14] Nicolas Bonifas and Philippe Baptiste. Nouvelles Bornes pour le RCPSP par Reformulation de Ressources Cumulatives. In *ROADEF-15ème congrès annuel de la Société française de recherche opérationnelle et d'aide à la décision*, 2014.

[BB17a] Philippe Baptiste and Nicolas Bonifas. Redundant cumulative constraints to compute preemptive bounds. *Discrete Applied Mathematics*, June 2017.

[BB17b] Nicolas Bonifas and Réda Bousseta. Boosting cumulative propagations with cumulative strengthening. In *ROADEF 2017*, 2017.

[BD04] Philippe Baptiste and Sophie Demassey. Tight LP Bounds for Resource Constrained Project Scheduling. *OR Spectrum*, 26(2):251–262, 2004.

[BK00] Peter Brucker and Sigrid Knust. A Linear Programming and Constraint Propagation-Based Lower Bound for the RCPSP. *European Journal of Operational Research*, 127(2):355–362, 2000.

[BK03] Peter Brucker and Sigrid Knust. Lower Bounds for Resource-Constrained Project Scheduling Problems. *European Journal of Operational Research*, 149(2):302–313, 2003.

[BK12] Peter Brucker and Sigrid Knust. *Complex Scheduling*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[Bon14] Nicolas Bonifas. Fast propagation for the Energy Reasoning. In *Doctoral Program of the 20th International Conference on Principles and Practice of Constraint Programming (CP)*, pages 16–22, 2014.

[Bon16] Nicolas Bonifas. A O (n^2 log (n)) propagation for the Energy Reasoning. In *Conference Paper, Roadef*, 2016.

[BPN01] Philippe Baptiste, Claude Le Pape, and Wim Nuijten. *Constraint-Based Scheduling: Applying Constraint Programming to Scheduling Problems*. Kluwer Academic Publishers, Norwell, MA, USA, 2001.

[BY98] Jean-Daniel Boissonnat and Mariette Yvinec. *Algorithmic Geometry*. Cambridge university press, 1998.

[CAdC10] François Clautiaux, Cláudio Alves, and José Valério de Carvalho. A Survey of Dual-Feasible and Superadditive Functions. *Annals of Operations Research*, 179(1):317–342, 2010.

[CAVT87] Nicos Christofides, R. Alvarez-Valdes, and J.M. Tamarit. Project scheduling with resource constraints: A branch and bound approach. *European Journal of Operational Research*, 29(3):262–273, June 1987.

[CN00] Jacques Carlier and Emmanuel Néron. An exact method for solving the multi-processor flow-shop. *RAIRO-Operations Research*, 34(1):1–25, 2000.

[CN03] Jacques Carlier and Emmanuel Néron. On linear lower bounds for the resource constrained project scheduling problem. *European Journal of Operational Research*, 149(2):314–324, 2003.

[CN07] Jacques Carlier and Emmanuel Néron. Computing Redundant Resources for the Resource Constrained Project Scheduling Problem. *European Journal of Operational Research*, 176(3):1452–1463, 2007.

[CO96] Amedeo Cesta and Angelo Oddi. Gaining Efficiency and Flexibility in the Simple Temporal Problem. In *Temporal Representation and Reasoning, 1996.(TIME'96), Proceedings., Third International Workshop on*, pages 45–50. IEEE, 1996.

[CP04] Jacques Carlier and Eric Pinson. Jackson's Pseudo-Preemptive Schedule and Cumulative Scheduling Problems. *Discrete Applied Mathematics*, 145(1):80–94, 2004.

[DH97] Erik L. Demeulemeester and Willy S. Herroelen. New Benchmark Results for the Resource-Constrained Project Scheduling Problem. *Management Science*, 43(11):1485–1492, November 1997.

[DKPV] Dürr, Christoph, Knust, Sigrid, Prot, Damien, and Vásquez, Óscar C. The Scheduling Zoo. http://www-desir.lip6.fr/~durrc/query/.

[DP13] Alban Derrien and Thierry Petit. The Energetic Reasoning Checker Revisited. *CP Doctoral Program 2013*, page 55, 2013.

[DP14] Alban Derrien and Thierry Petit. A New Characterization of Relevant Intervals for Energetic Reasoning. In *Principles and Practice of Constraint Programming*, pages 289–297. Springer, 2014.

[EL90] Jacques Erschler and Pierre Lopez. Energy-based approach for task scheduling under time and resources constraints. In *2nd international workshop on project management and scheduling*, pages 115–121, 1990.

[Ful62] Delbert R. Fulkerson. Expected Critical Path Lengths in PERT Networks. *Operations Research*, 10(6):808–817, December 1962.

[GJ75] M. Garey and D. Johnson. Complexity Results for Multiprocessor Scheduling under Resource Constraints. *SIAM Journal on Computing*, 4(4):397–411, December 1975.

[GLN05] Daniel Godard, Philippe Laborie, and Wim Nuijten. Randomized Large Neighborhood Search for Cumulative Scheduling. In *ICAPS*, volume 5, pages 81–89, 2005.

[GMS16] David Gerault, Marine Minier, and Christine Solnon. Constraint Programming Models for Chosen Key Differential Cryptanalysis. In *Principles and Practice of Constraint Programming*, pages 584–601. Springer, Cham, September 2016. DOI: 10.1007/978-3-319-44953-1_37.

[GR93] Andrew V. Goldberg and Tomasz Radzik. A Heuristic Improvement of the Bellman-Ford Algorithm. *Applied Mathematics Letters*, 6(3):3–6, 1993.

[Her06] Jeffrey W. Herrmann. A History of Production Scheduling. In Jeffrey W. Herrmann, editor, *Handbook of Production Scheduling*, number 89 in International Series in Operations Research & Management Science, pages 1–22. Springer US, 2006. DOI: 10.1007/0-387-33117-4_1.

[HK00] Sönke Hartmann and Rainer Kolisch. Experimental evaluation of state-of-the-art heuristics for the resource-constrained project scheduling problem. *European Journal of Operational Research*, 127(2):394–407, December 2000.

[IBM] IBM. IBM ILOG CPLEX CP Optimizer. http://www-01.ibm.com/software/commerce/optimization/cplex-cp-optimizer/.

[KALM11] Oumar Koné, Christian Artigues, Pierre Lopez, and Marcel Mongeau. Event-based MILP models for resource-constrained project scheduling problems. *Computers & Operations Research*, 38(1):3–13, January 2011.

[KHR+02] Henry Kautz, Eric Horvitz, Yongshao Ruan, Carla Gomes, and Bart Selman. Dynamic restart policies. *Aaai/iaai*, 97:674–681, 2002.

[KS97] Rainer Kolisch and Arno Sprecher. PSPLIB - a Project Scheduling Problem Library. *European journal of operational research*, 96(1):205–216, 1997.

[Lab03a] Philippe Laborie. Algorithms for Propagating Resource Constraints in AI Planning and Scheduling: Existing Approaches and New Results. *Artificial Intelligence*, 143(2):151–188, February 2003.

[Lab03b] Philippe Laborie. Resource Temporal Networks: Definition and Complexity. In *IJCAI*, pages 948–953, 2003.

[Lab05] Philippe Laborie. Complete MCS-Based Search: Application to Resource Constrained Project Scheduling. In *IJCAI*, pages 181–186, 2005.

[Lab09] Philippe Laborie. IBM ILOG CP Optimizer for Detailed Scheduling Illustrated on Three Problems. In *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, pages 148–162. Springer Berlin Heidelberg, 2009.

[LG07] Philippe Laborie and Daniel Godard. Self-Adapting Large Neighborhood Search: Application to Single-Mode Scheduling Problems. In *Proceedings MISTA-07, Paris*, pages 276–284, 2007.

[LMM02] Andrea Lodi, Silvano Martello, and Michele Monaci. Two-Dimensional Packing Problems: a Survey. *European journal of operational research*, 141(2):241–252, 2002.

[LR08] Philippe Laborie and Jérôme Rogerie. Reasoning with Conditional Time-Intervals. In *FLAIRS conference*, pages 555–560, 2008.

[LR14] Philippe Laborie and Jérôme Rogerie. Temporal Linear Relaxation in IBM ILOG CP Optimizer. *Journal of Scheduling*, pages 1–10, 2014.

[LRS+12] Philippe Laborie, Jérôme Rogerie, Paul Shaw, Petr Vilím, and Ferenc Katai. Interval-Based Language for Modeling Scheduling Problems: An Extension to Constraint Programming. In *Algebraic Modeling Systems*, pages 111–143. Springer Berlin Heidelberg, 2012.

[LRSV09] Philippe Laborie, Jerome Rogerie, Paul Shaw, and Petr Vilím. Reasoning with Conditional Time-Intervals. Part II: an Algebraical Model for Resources. In *FLAIRS conference*, 2009.

[LSZ93] Michael Luby, Alistair Sinclair, and David Zuckerman. Optimal speedup of Las Vegas algorithms. *Information Processing Letters*, 47(4):173–180, September 1993.

[McN59] Robert McNaughton. Scheduling with Deadlines and Loss Functions. *Management Science*, 6(1):1–12, 1959.

[MMRB98] Aristide Mingozzi, Vittorio Maniezzo, Salvatore Ricciardelli, and Lucio Bianco. An Exact Algorithm for the Resource-Constrained Project Scheduling Problem Based on a New Mathematical Formulation. *Management Science*, 44(5):714–729, May 1998.

[MSSU99] Rolf H. Möhring, Andreas S. Schulz, Frederik Stork, and Marc Uetz. Resource-Constrained Project Scheduling: Computing Lower Bounds by Solving Minimum Cut Problems. In *Algorithms-ESA'99*, pages 139–150. Springer, 1999.

[MSSU03] Rolf H. Möhring, Andreas S. Schulz, Frederik Stork, and Marc Uetz. Solving Project Scheduling Problems by Minimum Cut Computations. *Management Science*, 49(3):330–350, March 2003.

[MT90] Silvano Martello and Paolo Toth. Lower bounds and reduction procedures for the bin packing problem. *Discrete applied mathematics*, 28(1):59–70, 1990.

[MVH08] Luc Mercier and Pascal Van Hentenryck. Edge Finding for Cumulative Scheduling. *INFORMS Journal on Computing*, 20(1):143–153, 2008.

[NAB+06] Emmanuel Néron, Christian Artigues, Philippe Baptiste, Jacques Carlier, Jean Damay, Sophie Demassey, and Philippe Laborie. Lower Bounds for Resource Constrained Project Scheduling Problem. In *Perspectives in Modern Project Scheduling*, number 92 in International Series in Operations Research & Management Science, pages 167–204. Springer, 2006.

[Nui94] Wim P. M. Nuijten. *Time and Resource Constrained Scheduling: a Constraint Satisfaction Approach*. PhD thesis, Technische Universiteit Eindhoven, 1994.

[OQ13] Pierre Ouellet and Claude-Guy Quimper. Time-Table Extended-Edge-Finding for the Cumulative Constraint. In *Principles and Practice of Constraint Programming*, pages 562–577. Springer, 2013.

[OSC09] Olga Ohrimenko, Peter J. Stuckey, and Michael Codish. Propagation Via Lazy Clause Generation. *Constraints*, 14(3):357–391, 2009.

[Pap] Claude Le Pape. *Constraint-Based Scheduling: a Tutorial*.

[PCVG94] Claude Le Pape, Philippe Couronné, Didier Vergamini, and Vincent Gosselin. *Time-versus-Capacity Compromises in Project Scheduling*. 1994.

[Pin12] Michael L. Pinedo. *Scheduling*. Springer US, Boston, MA, 2012.

[PSCO04] Nicola Policella, Stephen F. Smith, Amedeo Cesta, and Angelo Oddi. Generating robust schedules through temporal flexibility. In *In Proceedings 14th International Conference on Automated Planning and Scheduling*, 2004.

[PWW69] A. Alan B. Pritsker, Lawrence J. Waiters, and Philip M. Wolfe. Multiproject Scheduling with Limited Resources: A Zero-One Programming Approach. *Management Science*, 16(1):93–108, September 1969.

[QS94] Maurice Queyranne and Andreas S. Schulz. Polyhedral Approaches to Machine Scheduling. Technical report, 1994.

[SAHL12] Gilles Simonin, Christian Artigues, Emmanuel Hebrard, and Pierre Lopez. Scheduling Scientific Experiments on the Rosetta/Philae Mission. In *Principles and Practice of Constraint Programming*, pages 23–37. Springer, Berlin, Heidelberg, 2012. DOI: 10.1007/978-3-642-33558-7_5.

[SFSW10] Andreas Schutt, Thibaut Feydy, Peter J. Stuckey, and Mark G. Wallace. Explaining the Cumulative Propagator. *Constraints*, 16(3):250–282, August 2010.

[Sha98] Paul Shaw. Using Constraint Programming and Local Search Methods to Solve Vehicle Routing Problems. In *Principles and Practice of Constraint Programming — CP98*, Lecture Notes in Computer Science, pages 417–431. Springer, Berlin, Heidelberg, October 1998.

[SW10] Andreas Schutt and Armin Wolf. A New O(n^2 log n) Not-First/Not-Last Pruning Algorithm for Cumulative Resource Constraints. In *Principles and Practice of Constraint Programming – CP 2010*, Lecture Notes in Computer Science, pages 445–459. Springer, September 2010.

[VBČ05] Petr Vilím, Roman Barták, and Ondřej Čepek. Extension of O(n log n) Filtering Algorithms for the Unary Resource Constraint to Optional Activities. *Constraints*, 10(4):403–425, 2005.

[VCLS15] Sascha Van Cauwelaert, Michele Lombardi, and Pierre Schaus. Understanding the Potential of Propagators. In Laurent Michel, editor, *Integration of AI and OR Techniques in Constraint Programming*, volume 9075, pages 427–436. Springer International Publishing, Cham, 2015. DOI: 10.1007/978-3-319-18008-3_29.

[VCLSo14] Sascha Van Cauwelaert, Michele Lombardi, Pierre Schaus, and others. Supervised learning to control energetic reasoning: Feasibility study. *Proceedings of the Doctoral Program CP2014*, 2014.

[Vil04] Petr Vilím. O(n log n) Filtering Algorithms for Unary Resource Constraint. In *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, volume 3011, pages 335–347, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

[Vil08] Petr Vilím. Filtering Algorithms for the Unary Resource Constraint. *Archives of Control Sciences*, 18:159–202, 2008.

[Vil09a] Petr Vilím. Edge Finding Filtering Algorithm for Discrete Cumulative Resources in O(kn log n). In *CP'09: Proceedings of the 15th international conference on Principles and practice of constraint programming*, pages 802–816, Berlin, Heidelberg, 2009. Springer.

[Vil09b] Petr Vilím. Max Energy Filtering Algorithm for Discrete Cumulative Resources. In *Integration of AI and OR techniques in constraint programming for combinatorial optimization problems*, pages 294–308. Springer, 2009.

[Vil11] Petr Vilím. Timetable Edge Finding Filtering Algorithm for Discrete Cumulative Resources. In Tobias Achterberg and J. Beck, editors, *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, volume 6697 of *Lecture Notes in Computer Science*, pages 230–245. Springer, 2011.

[VLS15] Petr Vilím, Philippe Laborie, and Paul Shaw. Failure-Directed Search for Constraint-Based Scheduling. In *Integration of AI and OR Techniques in Constraint Programming*, pages 437–453. Springer International Publishing, 2015.

Appendix: supplementary work

In parallel of preparing this PhD thesis in constraint-based scheduling, I continued working on topics I started during my Master's degree, namely on the geometrical structures of linear programming (polyhedra) and of integer linear programming (lattices).

This work resulted in two articles which have been prepared and published during my PhD.

The article "On Sub-Determinants and the Diameter of Polyhedra" gives a bound on the diameter of the constraint polyhedron of a linear program whose linear inequalities involve only integer coefficients. This bound is a function of the dimension and of the largest subdeterminant appearing in the constraint matrix. This is an extension of results known on the diameter of totally unimodular polyhedra.

The article "Short Paths on the Voronoi Graph and Closest Vector Problem with Preprocessing" significantly improves the complexity of the Micciancio-Voulgaris algorithm to solve the Closest Vector Problem with Preprocessing, in the field of Lattice Problems.

These two articles are included here for reference.

On sub-determinants and the diameter of polyhedra*

Nicolas Bonifas† Marco Di Summa‡ Friedrich Eisenbrand§ Nicolai Hähnle¶
Martin Niemeier‖

Abstract

We derive a new upper bound on the diameter of a polyhedron $P = \{x \in \mathbb{R}^n : Ax \leqslant b\}$, where $A \in \mathbb{Z}^{m\times n}$. The bound is polynomial in n and the largest absolute value of a sub-determinant of A, denoted by Δ. More precisely, we show that the diameter of P is bounded by $O\left(\Delta^2 n^4 \log n\Delta\right)$. If P is bounded, then we show that the diameter of P is at most $O\left(\Delta^2 n^{3.5} \log n\Delta\right)$.

For the special case in which A is a totally unimodular matrix, the bounds are $O\left(n^4 \log n\right)$ and $O\left(n^{3.5} \log n\right)$ respectively. This improves over the previous best bound of $O(m^{16} n^3 (\log mn)^3)$ due to Dyer and Frieze [5].

1 Introduction

One of the fundamental open problems in optimization and discrete geometry is the question whether the diameter of a polyhedron can be bounded by a polynomial in the dimension and the number of its defining inequalities. The problem is readily explained: A *polyhedron* is a set of the form $P = \{x \in \mathbb{R}^n : Ax \leqslant b\}$, where $A \in \mathbb{R}^{m\times n}$ is a matrix and $b \in \mathbb{R}^m$ is an m-dimensional vector. A *vertex* of P is a point $x^* \in P$ such that there exist n linearly independent rows of A whose corresponding inequalities of $Ax \leqslant b$ are satisfied by x^* with equality. Throughout this paper, we assume that the polyhedron P is *pointed*, i.e. it has vertices, which is equivalent to saying that the matrix A has full column-rank. Two different vertices x^* and y^* are *neighbors* if they are the endpoints of an *edge* of the polyhedron, i.e. there exist $n-1$ linearly independent rows of A whose corresponding inequalities of $Ax \leqslant b$ are satisfied with equality both by x^* and y^*. In this way, we obtain the undirected *polyhedral graph* with edges being pairs of neighboring vertices of P. This graph is connected. The *diameter* of P is the smallest natural number that bounds the length of a shortest path between any pair of vertices in this graph. The question is now as follows:

> Can the diameter of a polyhedron $P = \{x \in \mathbb{R}^n : Ax \leqslant b\}$ be bounded by a polynomial in m and n?

The belief in a positive answer to this question is called the *polynomial Hirsch conjecture*. Despite a lot of research effort during the last 50 years, the gap between lower and upper bounds on the diameter remains huge. While, when the dimension n is fixed, the diameter can be bounded by a linear function of m [14, 2], for the general case the best upper bound, due to Kalai and Kleitman [11],

*An extended abstract of this paper was presented at the 28-th annual ACM symposium on Computational Geometry (SOCG 12)

†LIX, École Polytechnique, Palaiseau and IBM, Gentilly (France). bonifas@lix.polytechnique.fr

‡Dipartimento di Matematica, Università di Padova (Italy). disumma@math.unipd.it

§Ecole Polytechnique Fédérale de Lausanne (Switzerland). friedrich.eisenbrand@epfl.ch

¶Technische Universität Berlin (Germany). haehnle@math.tu-berlin.de

‖Technische Universität Berlin (Germany). martin.niemeier@tu-berlin.de

is $m^{1+\log n}$. The best lower bound is of the form $(1+\varepsilon)\cdot m$ for some $\varepsilon > 0$ in fixed and sufficiently large dimension n. This is due to a celebrated result of Santos [19] who disproved the, until then longstanding, original *Hirsch conjecture* for polytopes. The Hirsch conjecture stated that the diameter of a bounded polyhedron[1] is at most $m-n$. Interestingly, this huge gap (polynomial versus quasi-polynomial) is also not closed in a very simple combinatorial abstraction of polyhedral graphs [6]. However, it was shown by Vershynin [20] that every polyhedron can be perturbed by a small random amount so that the expected diameter of the resulting polyhedron is bounded by a polynomial in m and n. See Kim and Santos [12] for a recent survey.

In light of the importance and apparent difficulty of the open question above, many researchers have shown that it can be answered in an affirmative way in some special cases. Naddef [17] proved that the Hirsch conjecture holds true for 0/1-polytopes. Orlin [18] provided a quadratic upper bound for flow-polytopes. Brightwell et al. [3] showed that the diameter of the transportation polytope is linear in m and n, and a similar result holds for the dual of a transportation polytope [1] and the axial 3-way transportation polytope [4].

The results on flow polytopes and classical transportation polytopes concern polyhedra defined by *totally unimodular matrices*, i.e., integer matrices whose sub-determinants are $0, \pm 1$. For such polyhedra Dyer and Frieze [5] had previously shown that the diameter is bounded by a polynomial in n and m. Their bound is $O(m^{16}n^3(\log mn)^3)$. Their result is also algorithmic: they show that there exists a randomized simplex-algorithm that solves linear programs defined by totally unimodular matrices in polynomial time.

Our main result is a generalization and considerable improvement of the diameter bound of Dyer and Frieze. We show that the diameter of a polyhedron $P = \{x \in \mathbb{R}^n \colon Ax \leqslant b\}$, with $A \in \mathbb{Z}^{m\times n}$ is bounded by $O\big(\Delta^2 n^4 \log n\Delta\big)$. Here, Δ denotes the largest absolute value of a *sub-determinant* of A. If P is bounded, i.e., a *polytope*, then we can show that the diameter of P is at most $O\big(\Delta^2 n^{3.5} \log n\Delta\big)$. To compare our bound with the one of Dyer and Frieze one has to set Δ above to one and obtains $O\big(n^4 \log n\big)$ and $O\big(n^{3.5} \log n\big)$ respectively. Notice that our bound is independent of m, i.e., the number of rows of A.

The proof method

Let u and v be two vertices of P. We estimate the maximum number of iterations of two breadth-first-search explorations of the polyhedral graph, one initiated at u, the other initiated at v, until a common vertex is discovered. The diameter of P is at most twice this number of iterations. The main idea in the analysis is to reason about the normal cones of vertices of P and to exploit a certain volume expansion property.

We can assume that $P = \{x \in \mathbb{R}^n \colon Ax \leqslant b\}$ is *non-degenerate*, i.e., each vertex has exactly n tight inequalities. This can be achieved by slightly perturbing the right-hand side vector b: in this way the diameter can only grow. We denote the polyhedral graph of P by $G_P = (V, E)$. Let $v \in V$ now be a vertex of P. The *normal cone* C_v of v is the set of all vectors $c \in \mathbb{R}^n$ such that v is an optimal solution of the linear program $\max\{c^T x \colon x \in \mathbb{R}^n,\ Ax \leqslant b\}$. The normal cone C_v of a vertex of v is a full-dimensional simplicial polyhedral cone. Two vertices v and v' are adjacent if and only if C_v and $C_{v'}$ share a facet. No two distinct normal cones share an interior point. Furthermore, if P is a polytope, then the union of the normal cones of vertices of P is the complete space $\mathbb{R}^n$.

We now define the *volume* of a set $U \subseteq V$ of vertices as the volume of the union of the normal cones of U intersected with the *unit ball* $B_n = \{x \in \mathbb{R}^n \colon \|x\|_2 \leqslant 1\}$, i.e.,

$$\operatorname{vol}(U) := \operatorname{vol}\left(\bigcup_{v\in U} C_v \cap B_n\right).$$

[1] A counterexample to the same conjecture for unbounded polyhedra was found in 1967 by Klee and Walkup [13].

Consider an iteration of breadth-first-search. Let $I \subseteq V$ be the set of vertices that have been discovered so far. Breadth-first-search will next discover the neighborhood of I, which we denote by $\mathcal{N}(I)$.

Together with the integrality of A, the bound Δ on the subdeterminants guarantees that the angle between one facet of a normal cone C_v and the opposite ray is not too small. We combine this fact, which we formalize in Lemma 3, with an isoperimetric inequality to show that the volume of $\mathcal{N}(I)$ is large relative to the volume of I.

Lemma 1. *Let $P = \{x \in \mathbb{R}^n : Ax \leqslant b\}$ be a polytope where all sub-determinants of $A \in \mathbb{Z}^{m\times n}$ are bounded by Δ in absolute value and let $I \subseteq V$ be a set of vertices of G_P with $\mathrm{vol}(I) \leqslant (1/2)\cdot \mathrm{vol}(B_n)$. Then the volume of the neighborhood of I is at least*

$$\mathrm{vol}(\mathcal{N}(I)) \geqslant \sqrt{\frac{2}{\pi}}\frac{1}{\Delta^2 n^{2.5}} \cdot \mathrm{vol}(I).$$

We provide the proof of this lemma in the next section. Our diameter bound for polytopes is an easy consequence:

Theorem 2. *Let $P = \{x \in \mathbb{R}^n : Ax \leqslant b\}$ be a polytope where all subdeterminants of $A \in \mathbb{Z}^{m\times n}$ are bounded by Δ in absolute value. The diameter of P is bounded by $O\left(\Delta^2 n^{3.5} \log n\Delta\right)$.*

Proof. We estimate the maximum number of iterations of breadth-first-search until the total volume of the discovered vertices exceeds $(1/2)\cdot \mathrm{vol}(B_n)$. This is an upper bound on the aforementioned maximum number of iterations of two breadth-first-search explorations until a common vertex is discovered.

Suppose we start at vertex v and let I_j be the vertices that have been discovered during the first j iterations. We have $I_o = \{v\}$. If $j \geqslant 1$ and $\mathrm{vol}(I_{j-1}) \leqslant (1/2)\cdot \mathrm{vol}(B_n)$ we have by Lemma 1

$$\begin{aligned}\mathrm{vol}(I_j) &\geqslant \left(1 + \sqrt{\frac{2}{\pi}}\frac{1}{\Delta^2 n^{2.5}}\right)\mathrm{vol}(I_{j-1}) \\ &\geqslant \left(1 + \sqrt{\frac{2}{\pi}}\frac{1}{\Delta^2 n^{2.5}}\right)^j \mathrm{vol}(I_0).\end{aligned}$$

The condition $\mathrm{vol}(I_j) \leqslant (1/2)\cdot \mathrm{vol}(B_n)$ implies

$$\left(1 + \frac{1}{\sqrt{\frac{\pi}{2}}\Delta^2 n^{2.5}}\right)^j \mathrm{vol}(I_0) \leqslant 2^n.$$

This is equivalent to

$$j \cdot \ln\left(1 + \frac{1}{\sqrt{\frac{\pi}{2}}\Delta^2 n^{2.5}}\right) \leqslant \ln(2^n/\mathrm{vol}(I_0)).$$

For $0 \leqslant x \leqslant 1$ one has $\ln(1+x) \geqslant x/2$ and thus the inequality above implies

$$j \leqslant \sqrt{2\pi}\Delta^2 n^{2.5} \cdot \ln(2^n/\mathrm{vol}(I_0)). \tag{1}$$

To finish the proof we need a lower bound on $\mathrm{vol}(I_0)$, i.e., the n-dimensional volume of the set $C_v \cap B_n$. The normal cone C_v contains the full-dimensional simplex spanned by 0 and the n row-vectors $a_{i_1}, \ldots, a_{i_n}$ of A that correspond to the inequalities of $Ax \leqslant b$ that are tight at v. Since A is integral, the volume of this simplex is at least $1/n!$. Furthermore, if this simplex is scaled by $1/\max\{\|a_{i_k}\| : k = 1,\ldots,n\}$, then it is contained in the unit ball. Since each component of A is itself a sub-determinant, one has $\max\{\|a_{i_k}\| : k = 1,\ldots,n\} \leqslant \sqrt{n}\Delta$ and thus $\mathrm{vol}(I_0) \geqslant 1/(n!\cdot n^{n/2}\Delta^n)$. It follows that (1) implies $j = O\left(\Delta^2 n^{3.5}\log n\Delta\right)$. □

Remarks. The result of Dyer and Frieze [5] is also based on analyzing expansion properties via isoperimetric inequalities. It is our choice of normal cones as the natural geometric representation, and the fact that we only ask for volume expansion instead of expansion of the graph itself, that allows us to get a better bound. Expansion properties of the graphs of general classes of polytopes have also been studied elsewhere in the literature, e.g. [10, 9].

Organization of the paper

The next section is devoted to a proof of the volume-expansion property, i.e., Lemma 1. The main tool that is used here is a classical *isoperimetric inequality* that states that among measurable subsets of a sphere with fixed volume, spherical caps have the smallest circumference. Section 3 deals with unbounded polyhedra. Compared to the case of polytopes, the problem that arises here is the fact that the union of the normal cones is not the complete space $\mathbb{R}^n$. To tackle this case, we rely on an isoperimetric inequality of Lovász and Simonovits [15]. Finally, we discuss how our bound can be further generalized. In fact, not all sub-determinants of A need to be at most Δ but merely the entries of A and the $(n-1)$-dimensional sub-determinants have to be bounded by Δ, which yields a slightly stronger result.

2 Volume expansion

This section is devoted to a proof of Lemma 1. Throughout this section, we assume that $P = \{x \in \mathbb{R}^n : Ax \leqslant b\}$ is a polytope. We begin with some useful notation. A (not necessarily convex) *cone* is a subset of $\mathbb{R}^n$ that is closed under the multiplication with non-negative scalars. The intersection of a cone with the unit ball B_n is called a *spherical cone*. Recall that C_v denotes the normal cone of the vertex v of P. We denote the spherical cone $C_v \cap B_n$ by S_v and, for a subset $U \subseteq V$, the spherical cone $\bigcup_{v\in U} S_v$ by S_U. Our goal is to show that the following inequality holds for each $I \subseteq V$ with $\mathrm{vol}(S_I) \leqslant \frac{1}{2}\mathrm{vol}(B_n)$:

$$\mathrm{vol}(S_{\mathcal{N}(I)}) \geqslant \sqrt{\frac{2}{\pi}} \frac{1}{\Delta^2 n^{2.5}} \cdot \mathrm{vol}(S_I). \tag{2}$$

Recall that two vertices are adjacent in G_P if and only if their normal cones have a common facet. This means that the neighbors of I are those vertices u for which S_u has a facet which is part of the surface of the spherical cone S_I. In an iteration of breadth-first-search we thus augment the set of discovered vertices I by those vertices u that can "dock" on S_I via a common facet. We call the $(n-1)$-dimensional volume of the surface of a spherical cone S that is not on the sphere, the *dockable surface* $D(S)$, see Figure 1.

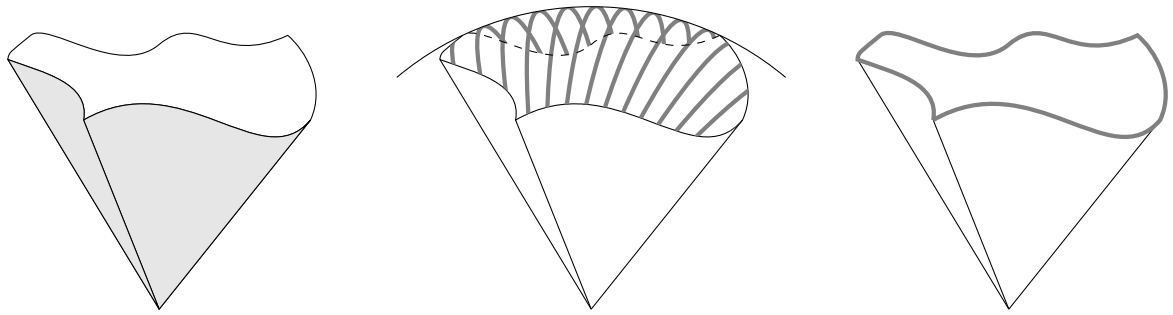
The *base* of S is the intersection of S with the unit sphere. We denote the area of the base by $B(S)$. By *area* we mean the $(n-1)$-dimensional measure of some surface. Furthermore, $L(S)$ denotes the length of the relative boundary of the base of S. We use the term *length* to denote the measure of an $(n-2)$-dimensional volume, see Figure 1.

Given any spherical cone S in the unit ball, the following well-known relations follow from basic integration:

$$\mathrm{vol}(S) = \frac{B(S)}{n}, \quad D(S) = \frac{L(S)}{n-1}. \tag{3}$$

To obtain the volume expansion relation (2) we need to bound the dockable surface of a spherical cone from below by its volume and, for a simplicial spherical cone, we need an upper bound on the dockable surface by its volume. More precisely, we show that for every simplicial spherical cone S_v one has

$$\frac{D(S_v)}{\mathrm{vol}(S_v)} \leqslant \Delta^2 n^3 \tag{4}$$



(a) Dockable surface of S. (b) Base of S. (c) Relative boundary of the base of S.

Figure 1: Illustration of $D(S)$, $B(S)$ and $L(S)$.

and for any spherical cone one has

$$\frac{D(S)}{\mathrm{vol}(S)} \geqslant \sqrt{\frac{2n}{\pi}}. \tag{5}$$

Once inequalities (4) and (5) are derived, the bound (2) can be obtained as follows. All of the dockable surface of S_I must be "consumed" by the neighbors of I. Using (5) one has thus

$$\sum_{v \in \mathcal{N}(I)} D(S_v) \geqslant D(S_I) \geqslant \sqrt{\frac{2n}{\pi}} \cdot \mathrm{vol}(S_I). \tag{6}$$

On the other hand, (4) implies

$$\sum_{v \in \mathcal{N}(I)} D(S_v) \leqslant \Delta^2 n^3 \cdot \sum_{v \in \mathcal{N}(I)} \mathrm{vol}(S_v) = \Delta^2 n^3 \cdot \mathrm{vol}(S_{\mathcal{N}(I)}). \tag{7}$$

These last two inequalities imply inequality (2). The remainder of this section is devoted to proving (4) and (5).

2.1 Area to volume ratio of a spherical simplicial cone

We will first derive inequality (4).

Lemma 3. *Let v be a vertex of P. One has*

$$\frac{D(S_v)}{\mathrm{vol}(S_v)} \leqslant \Delta^2 n^3.$$

Proof. Let F be a facet of a spherical cone S_v. Let y be the vertex of S_v not contained in F. Let Q denote the convex hull of F and y (see Figure 2). We have $Q \subseteq S_v$ because S_v is convex. Moreover, if h_F is the Euclidean distance of y from the hyperplane containing F, then

$$\mathrm{vol}(S_v) \geqslant \mathrm{vol}(Q) = \frac{\mathrm{area}(F) \cdot h_F}{n}.$$

Summing over the facets of S_v, we find

$$\frac{D(S_v)}{\mathrm{vol}(S_v)} = \sum_{\text{facet } F} \frac{\mathrm{area}(F)}{\mathrm{vol}(S_v)} \leqslant n \cdot \sum_{\text{facet } F} \frac{1}{h_F}. \tag{8}$$

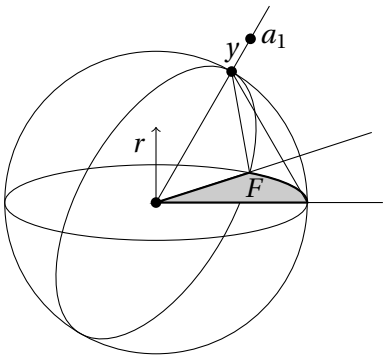


Figure 2: Proof of Lemma 3.

It remains to provide a lower bound on h_F. Let $a_1, \ldots, a_n$ be the row-vectors of A defining the extreme rays of the normal cone of v, and let A_v be the non-singular matrix whose rows are $a_1, \ldots, a_n$. Furthermore, suppose that the vertex y lies on the ray generated by a_1. Let H be the hyperplane generated by $a_2, \ldots, a_n$. The distance $d(y, H)$ of y to H is equal to $d(a_1, H)/\|a_1\|$. Let $b_1, \ldots, b_n$ be the columns of the adjugate of A_v. The column-vector b_1 is integral and each component of b_1 is bounded by Δ. Furthermore b_1 is orthogonal to each of $a_2, \ldots, a_n$. Thus $d(a_1, H)$ is the length of the projection of a_1 to b_1, which is $|\langle a_1, b_1\rangle|/\|b_1\| \geqslant 1/(\sqrt{n}\cdot\Delta)$, since a_1 and b_1 are integral. Thus

$$h_F = d(y, H) \geqslant \frac{1}{n\Delta^2}.$$

Plugging this into (8) completes the proof. □

2.2 An isoperimetric inequality for spherical cones

We now derive the lower bound (5) on the area to volume ratio for a general spherical cone. To do that, we assume that the spherical cone has the least favorable shape for the area to volume ratio and derive the inequality for cones of this shape. Here one uses classical isoperimetric inequalities. The basic isoperimetric inequality states that the measurable subset of $\mathbb{R}^n$ with a prescribed volume and minimal area is the ball of this volume. In this paper, we need Lévy's isoperimetric inequality, see e.g. [8, Theorem 2.1], which can be seen as an analogous result for spheres: it states that a measurable subset of the sphere of prescribed area and minimal boundary is a spherical cap.

A spherical cone S is a *cone of revolution* if there exist a vector v and an angle $0 < \theta \leqslant \pi/2$ such that S is the set of vectors in the unit ball that form an angle of at most θ with v:

$$S = \left\{ x \in B_n : \frac{v^T x}{\|v\|\|x\|} \geqslant \cos\theta \right\}.$$

Note that a spherical cone is a cone of revolution if and only if its base is a spherical cap. We also observe that two spherical cones of revolution, defined by two different vectors but by the same angle, are always congruent. Therefore, in the following we will only specify the angle of a cone of revolution.

Lemma 4. *The spherical cone of given volume with minimum lateral surface is a cone of revolution.*

Proof. By the first equation of (3), every spherical cone of volume V intersects the unit sphere in a surface of area nV. Furthermore, by the second equation of (3), the length of the boundary of this surface is proportional to the area of the lateral surface of the cone. Then the problem of finding the spherical cone of volume V with the minimum lateral surface can be rephrased as follows: Find a surface of area nV on the unit sphere having the boundary of minimum length. By Lévy's isoperimetric

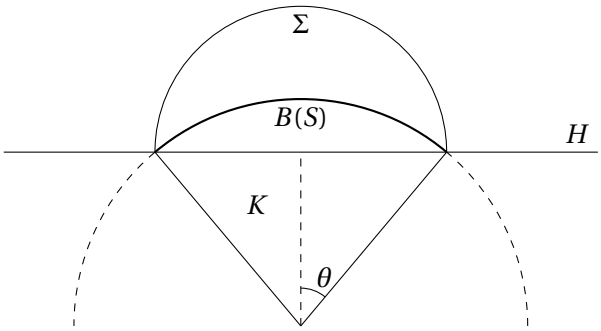


Figure 3: Proof of Lemma 5.

inequality for spheres, the optimal shape for such a surface is a spherical cap. As observed above, this corresponds to a cone of revolution. □

Lemma 5. *Let S be a spherical cone of revolution of angle $0 < \theta \leqslant \pi/2$. Then*

$$\frac{D(S)}{\mathrm{vol}(S)} \geqslant \sqrt{\frac{2n}{\pi}}.$$

Proof. Using (3), we have to show that

$$\frac{L(S)}{B(S)} \geqslant \sqrt{\frac{2}{\pi}}\frac{n-1}{\sqrt{n}}. \tag{9}$$

This is done in two steps. We first prove that this ratio is minimal for S being the half-ball, i.e., $\theta = \pi/2$. Then we show that $\frac{L(S)}{B(S)} \geqslant \sqrt{\frac{2}{\pi}}\frac{n-1}{\sqrt{n}}$ holds for the half-ball.

Let H be the hyperplane containing the boundary of the base of S. Then H divides S into two parts: a truncated cone K and the convex hull of a spherical cap. The radius r of the base of K is bounded by one.

Consider now the half-ball that contains $B(S)$ and that has $H \cap B_n$ as its flat-surface, see Figure 3, and let Σ denote the area of the corresponding half-sphere. One has $B(S) \leqslant \Sigma$ and thus

$$\frac{L(S)}{B(S)} \geqslant \frac{L(S)}{\Sigma}.$$

Now Σ and $L(S)$ are the surface of an $(n-1)$-dimensional half-sphere of radius r and the length of its boundary respectively. If we scale this half-sphere by a factor of $1/r$, we obtain the unit half-ball and its length respectively. Since scaling by a factor of $1/r$ increases areas by a factor of $1/r^{n-1}$ and lengths by a factor of $1/r^{n-2}$, we have that $\frac{L(S)}{\Sigma}$ is at least the length of the unit-half-ball divided by the area of the base of the half-ball.

Suppose now that S is the half-unit-ball. We show that the inequality $L(S)/B(S) \geqslant \sqrt{\frac{2}{\pi}}\frac{n-1}{\sqrt{n}}$ holds. The base of S is a half unit sphere and $L(S)$ is the length of the boundary of a unit ball of dimension $n-1$. Thus

$$B(S) = \frac{n}{2}\frac{\pi^{n/2}}{\Gamma\left(\frac{n}{2}+1\right)}, \quad L(S) = \frac{(n-1)\pi^{(n-1)/2}}{\Gamma\left(\frac{n-1}{2}+1\right)},$$

where Γ is the well-known Gamma function. Using the fact that $\Gamma(x+1/2)/\Gamma(x) \geqslant \sqrt{x-\frac{1}{4}}$ for all $x > \frac{1}{4}$ (see, e.g., [16]), one easily verifies that

$$\Gamma\left(\frac{n}{2}+1\right) \geqslant \sqrt{\frac{n}{2}} \cdot \Gamma\left(\frac{n-1}{2}+1\right).$$

It follows that

$$\frac{L(S)}{B(S)} = \frac{2}{\sqrt{\pi}} \frac{n-1}{n} \frac{\Gamma\left(\frac{n}{2}+1\right)}{\Gamma\left(\frac{n-1}{2}+1\right)} \geqslant \sqrt{\frac{2}{\pi}} \cdot \frac{n-1}{\sqrt{n}}.$$

□

Finally we are now ready to consider the case of an arbitrary spherical cone.

Lemma 6. *Let S be a (not necessarily convex) spherical cone with* $\mathrm{vol}(S) \leqslant \frac{1}{2}\mathrm{vol}(B_n)$. *Then*

$$\frac{D(S)}{\mathrm{vol}(S)} \geqslant \sqrt{\frac{2n}{\pi}}.$$

Proof. Let S^* be a spherical cone of revolution with the same volume as S. By Lemma 4, $D(S) \geqslant D(S^*)$. Now, using Lemma 5 one has

$$\frac{D(S)}{\mathrm{vol}(S)} \geqslant \frac{D(S^*)}{\mathrm{vol}(S^*)} \geqslant \sqrt{\frac{2n}{\pi}}.$$

□

This was the final step in the proof of Lemma 1 and thus we have also proved Theorem 2, our main result on polytopes. The next section is devoted to unbounded polyhedra.

3 The case of an unbounded polyhedron

If the polyhedron P is unbounded, then the union of the normal cones of all vertices of P forms a proper subset K' of $\mathbb{R}^n$: namely, K' is the set of objective functions c for which the linear program $\max\{c^T x : x \in P\}$ has finite optimum. Similarly, the set $K' \cap B_n$ is a proper subset of B_n. Then, given the union of the spherical cones that have already been discovered by the breadth-first-search (we denote this set by S), we should redefine the dockable surface of S as that part of the lateral surface of S that is shared by some neighboring cones. In other words, we should exclude the part lying on the boundary of $K' \cap B_n$. However, this implies that the result of Lemma 6 does not hold in this context.

To overcome this difficulty, we make use of the Lovász-Simonovits inequality, which we now recall. Below we use notation $d(X,Y)$ to indicate the Euclidean distance between two subsets $X, Y \subseteq \mathbb{R}^n$, i.e., $d(X,Y) = \inf\{\|x-y\| : x \in X, y \in Y\}$. Also, $[x,y]$ denotes the segment connecting two points $x, y \in \mathbb{R}^n$ (see Figure 4).

Theorem 7. *[15] Let $K \subseteq \mathbb{R}^n$ be a convex compact set, $0 < \varepsilon < 1$ and (K_1, K_2, K_3) be a partition of K into three measurable sets such that*

$$\forall x, y \in K, \quad d([x,y] \cap K_1, [x,y] \cap K_2) \geqslant \varepsilon \cdot \|x-y\|. \tag{10}$$

Then

$$\mathrm{vol}(K_3) \geqslant \frac{2\varepsilon}{1-\varepsilon} \min\left(\mathrm{vol}(K_1), \mathrm{vol}(K_2)\right).$$

We now illustrate how the above result can be used in our context. Let $K = K' \cap B_n$ and observe that K is a convex and compact set. Let $S \subseteq K$ be the union of the spherical cones that have already been discovered by the breadth-first-search. We define the dockable surface of S as that part of the lateral surface of S that is disjoint from the boundary of K. We denote by $D'(S)$ the area of the dockable surface of S. We can prove the following analogue of Lemma 6:

Lemma 8. *If* $\mathrm{vol}(S) \leqslant \frac{1}{2}\mathrm{vol}(K)$, *then* $D'(S) \geqslant \mathrm{vol}(S)$.

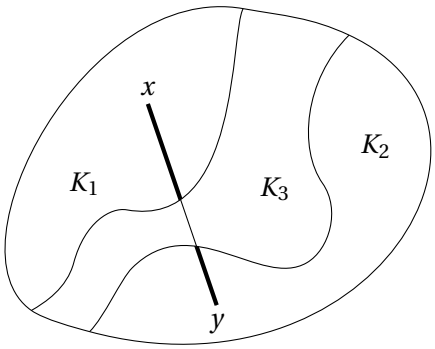


Figure 4: Illustration of the Lovász-Simonovits inequality.

Proof. Let F denote the dockable surface of S (thus $D'(S)$ is the area of F). For every $\varepsilon > 0$ we define

$$\begin{aligned} K_{3,\varepsilon} &= (F + \varepsilon B_n) \cap K, \\ K_{1,\varepsilon} &= S \setminus K_{3,\varepsilon}, \\ K_{2,\varepsilon} &= K \setminus (K_{1,\varepsilon} \cup K_{3,\varepsilon}), \end{aligned}$$

where $X + Y$ denotes the Minkowski sum of two subsets $X, Y \in \mathbb{R}^n$, i.e., $X + Y = \{x + y : x \in X, y \in Y\}$. Clearly $(K_{1,\varepsilon}, K_{2,\varepsilon}, K_{3,\varepsilon})$ is a partition of K into three measurable sets. Furthermore, condition (10) is satisfied. Thus Theorem 7 implies that

$$\frac{\mathrm{vol}(K_{3,\varepsilon})}{2\varepsilon} \geqslant \frac{1}{1-\varepsilon} \min\big(\mathrm{vol}(K_{1,\varepsilon}), \mathrm{vol}(K_{2,\varepsilon})\big).$$

We observe that

$$\begin{aligned} \mathrm{vol}(K_{2,\varepsilon}) &\geqslant \mathrm{vol}(K \setminus S) - \mathrm{vol}(K_{3,\varepsilon}) \\ &\geqslant \mathrm{vol}(S) - \mathrm{vol}(K_{3,\varepsilon}) \\ &\geqslant \mathrm{vol}(K_{1,\varepsilon}) - \mathrm{vol}(K_{3,\varepsilon}). \end{aligned}$$

Combining those two inequalities, we find

$$\frac{\mathrm{vol}(F + \varepsilon B_n)}{2\varepsilon} \geqslant \frac{\mathrm{vol}(K_{3,\varepsilon})}{2\varepsilon} \geqslant \frac{1}{1-\varepsilon}(\mathrm{vol}(K_{1,\varepsilon}) - \mathrm{vol}(K_{3,\varepsilon})). \tag{11}$$

By a well-known result in geometry (see, e.g., [7],) as ε tends to 0 the left-hand side of (11) tends to the area of F, which is precisely the dockable surface $D'(S)$. Moreover, as ε tends to 0, $\mathrm{vol}(K_{3,\varepsilon})$ tends to 0 and $\mathrm{vol}(K_{1,\varepsilon})$ tends to $\mathrm{vol}(S)$. We conclude that $D'(S) \geqslant \mathrm{vol}(S)$. □

Following the same approach as that used for the case of a polytope, one can show the following result for polyhedra.

Theorem 9. *Let $P = \{x \in \mathbb{R}^n : Ax \leqslant b\}$ be a polyhedron, where all sub-determinants of $A \in \mathbb{Z}^{m \times n}$ are bounded by Δ in absolute value. Then the diameter of P is bounded by $O\big(\Delta^2 n^4 \log n\Delta\big)$. In particular, if A is totally unimodular, then the diameter of P is bounded by $O(n^4 \log n)$.*

Remark

For simplicity, we have assumed that a bound Δ was given for the absolute value of all sub-determinants of A. However, our proof only uses the fact the the sub-determinants of size 1 (i.e., the entries of the matrix) and $n-1$ are bounded. Calling Δ_1 (resp. Δ_{n-1}) the bound on the absolute value of the entries of A (resp. on the sub-determinants of A of size $n-1$), one easily verifies that all the results discussed above remain essentially unchanged, except that the statement of Lemma 3 becomes

$$\frac{D(S_v)}{\mathrm{vol}(S_v)} \leqslant \Delta_1 \Delta_{n-1} n^3$$

and the lower bound on $\mathrm{vol}(I_0)$ becomes

$$\mathrm{vol}(I_0) \geqslant \frac{1}{n! n^{n/2} \Delta_1^n}.$$

This implies the following strengthened result:

Theorem 10. *Let $P = \{x \in \mathbb{R}^n : Ax \leqslant b\}$ be a polyhedron, where the entries of A (respectively the sub-determinants of A of size $n-1$) are bounded in absolute value by Δ_1 (respectively Δ_{n-1}). Then the diameter of P is bounded by $O\left(\Delta_1 \Delta_{n-1} n^4 \log n \Delta_1\right)$. Moreover, if P is a polytope, its diameter is bounded by $O\left(\Delta_1 \Delta_{n-1} n^{3.5} \log n \Delta_1\right)$.*

4 Acknowledgements

This work was carried out while all authors were at EPFL (École Polytechnique Fédérale de Lausanne), Switzerland. The authors acknowledge support from the DFG Focus Program 1307 within the project "Algorithm Engineering for Real-time Scheduling and Routing" and from the Swiss National Science Foundation within the project "Set-partitioning integer programs and integrality gaps".

References

[1] M. L. Balinski. The Hirsch conjecture for dual transportation polyhedra. *Math. Oper. Res.*, 9(4):629–633, 1984.

[2] D. Barnette. An upper bound for the diameter of a polytope. *Discrete Math.*, 10:9–13, 1974.

[3] G. Brightwell, J. van den Heuvel, and L. Stougie. A linear bound on the diameter of the transportation polytope. *Combinatorica*, 26(2):133–139, 2006.

[4] J. A. De Loera, E. D. Kim, S. Onn, and F. Santos. Graphs of transportation polytopes. *J. Combin. Theory Ser. A*, 116(8):1306–1325, 2009.

[5] M. Dyer and A. Frieze. Random walks, totally unimodular matrices, and a randomised dual simplex algorithm. *Math. Program.*, 64(1, Ser. A):1–16, 1994.

[6] F. Eisenbrand, N. Hähnle, A. Razborov, and T. Rothvoß. Diameter of polyhedra: Limits of abstraction. *Math. Oper. Res.*, 35(4):786–794, 2010.

[7] H. Federer. *Geometric Measure Theory*. Springer, 1969.

[8] T. Figiel, J. Lindenstrauss, and V. Milman. The dimension of almost spherical sections of convex bodies. *Acta Math.*, 139(1):53–94, 1977.

[9] V. Kaibel. On the expansion of graphs of 0/1-polytopes. In *The sharpest cut*, MPS/SIAM Ser. Optim., pages 199–216. SIAM, Philadelphia, PA, 2004.

[10] G. Kalai. The diameter of graphs of convex polytopes and f-vector theory. In *Applied geometry and discrete mathematics*, volume 4 of *DIMACS Ser. Discrete Math. Theoret. Comput. Sci.*, pages 387–411. Amer. Math. Soc., Providence, RI, 1991.

[11] G. Kalai and D. J. Kleitman. A quasi-polynomial bound for the diameter of graphs of polyhedra. *Bull. Amer. Math. Soc. (N.S.)*, 26(2):315–316, 1992.

[12] E. D. Kim and F. Santos. An update on the Hirsch conjecture. *Jahresber. Dtsch. Math.-Ver.*, 112(2):73–98, 2010.

[13] V. Klee and D. W. Walkup. The d-step conjecture for polyhedra of dimension $d < 6$. *Acta Math.*, 133:53–78, 1967.

[14] D. G. Larman. Paths of polytopes. *Proc. London Math. Soc. (3)*, 20:161–178, 1970.

[15] L. Lovász and M. Simonovits. Random walks in a convex body and an improved volume algorithm. *Rand. Struct. Algor.*, 4(4):359–412, 1993.

[16] M. Merkle. Logarithmic convexity and inequalities for the gamma function. *J. Math. Anal. Appl.*, 203(2):369–380, 1996.

[17] D. Naddef. The Hirsch conjecture is true for (0,1)-polytopes. *Math. Program.*, 45:109–110, 1989.

[18] J. B. Orlin. A polynomial time primal network simplex algorithm for minimum cost flows. *Math. Program.*, 78(2, Ser. B):109–129, 1997.

[19] F. Santos. A counterexample to the Hirsch conjecture. 2010. arXiv:1006.2814v1 [math.CO]. To appear in *Ann. Math.*

[20] R. Vershynin. Beyond Hirsch conjecture: walks on random polytopes and smoothed complexity of the simplex method. *SIAM J. Comput.*, 39(2):646–678, 2009.

Short Paths on the Voronoi Graph and Closest Vector Problem with Preprocessing

Daniel Dadush* Nicolas Bonifas†

Abstract

Improving on the Voronoi cell based techniques of [28, 24], we give a *Las Vegas* $\widetilde{O}(2^n)$ expected time and space algorithm for CVPP (the preprocessing version of the Closest Vector Problem, CVP). This improves on the $\widetilde{O}(4^n)$ deterministic runtime of the Micciancio Voulgaris algorithm [24] (henceforth MV) for CVPP [1] at the cost of a polynomial amount of randomness (which only affects runtime, not correctness).

As in MV, our algorithm proceeds by computing a short path on the Voronoi graph of the lattice, where lattice points are adjacent if their Voronoi cells share a common facet, from the origin to a closest lattice vector. Our main technical contribution is a randomized procedure that given the Voronoi relevant vectors of a lattice – the lattice vectors inducing facets of the Voronoi cell – as preprocessing and any "close enough" lattice point to the target, computes a path to a closest lattice vector of expected polynomial size. This improves on the $\widetilde{O}(2^n)$ path length given by the MV algorithm. Furthermore, as in MV, each edge of the path can be computed using a single iteration over the Voronoi relevant vectors.

As a byproduct of our work, we also give an optimal relationship between geometric and path distance on the Voronoi graph, which we believe to be of independent interest.

Keywords. Closest Vector Problem, Lattice Problems, Convex Geometry.

*Department of Computer Science, New York University, New York (USA). dadush@cs.nyu.edu

†LIX, École Polytechnique, Palaiseau and IBM, Gentilly (France). nicolas.bonifas@polytechnique.edu

[1] The MV algorithm also solves CVP, as the preprocessing can be computed in the same time bound.

1 Introduction

An n dimensional lattice $\mathcal{L}$ in $\mathbb{R}^n$ is defined as all integer combinations of some basis $B = (\mathbf{b}_1, \ldots, \mathbf{b}_n)$ of $\mathbb{R}^n$. The most fundamental computational problems on lattices are the Shortest and Closest Vector Problems, which we denote by SVP and CVP respectively. Given a basis $B \in \mathbb{R}^{n \times n}$ of $\mathcal{L}$, the SVP is to compute $\mathbf{y} \in \mathcal{L} \setminus \{\mathbf{0}\}$ minimizing $\|\mathbf{y}\|_2$, and the CVP is, given an additional target $\mathbf{t} \in \mathbb{R}^n$, to compute a vector $\mathbf{y} \in \mathcal{L}$ minimizing $\|\mathbf{t} - \mathbf{y}\|_2$ [2].

The study of the algorithms and complexity of lattice problems has yielded many fundamental results in Computer Science and other fields over the last three decades. Lattice techniques were introduced to factor polynomials with rational coefficients [20] and to show the polynomial solvability of integer programs with a fixed number of integer variables [20, 21]. It has been used as a cryptanalytic tool for breaking the security of knapsack crypto schemes [19], and in coding theory for developing structured codes [8] and asymptotically optimal codes for power-constrained additive white Gaussian noise (AWGN) channels [10]. Most recently, the security of powerful cryptographic primitives such as fully homomorphic encryption [12, 13, 6] have been based on the worst case hardness of lattice problems.

The Closest Vector Problem with Preprocessing. In CVP applications, a common setup is the need to solve many CVP queries over the same lattice but with varying targets. This is the case in the context of coding over a Gaussian noise channel, a fundamental channel model in wireless communication theory. Lattice codes, where the codewords correspond to a subset of lattice points, asymptotically achieve the AWGN channel capacity (for fixed transmission power), and maximum likelihood decoding for a noisy codeword corresponds (almost) exactly to a CVP query on the coding lattice. In the context of lattice based public key encryption, in most cases the decryption routine can be interpreted as solving an approximate (decisional) CVP over a public lattice, where the encrypted bit is 0 if the point is close and 1 if it is far.

CVP algorithms in this setting (and in general), often naturally break into a preprocessing phase, where useful information about the lattice is computed (i.e. short lattice vectors, a short basis, important sublattices, etc.), and a query / search phase, where the computed advice is used to answer CVP queries quickly. Since the advice computed during preprocessing is used across all CVP queries, if the number of CVP queries is large the work done in the preprocessing phase can be effectively "amortized out". This motivates the definition of the Closest Vector Problem with Preprocessing (CVPP), where we fix an n dimensional lattice $\mathcal{L}$ and measure only the complexity of answering CVP queries on $\mathcal{L}$ after the preprocessing phase has been completed (crucially, the preprocessing is done before the CVP queries are known). To avoid trivial solutions to this problem, i.e. not allowing the preprocessing phase to compute a table containing all CVP solutions, we restrict the amount of space (as a function of the encoding size of the input lattice basis) needed to store the preprocessing advice.

Complexity. While the ability to preprocess the lattice is very powerful, it was shown in [25] that CVPP is NP-hard when the size of the preprocessing advice is polynomial. Subsequently, approximation hardness for the gap version of CVPP (i.e. approximately deciding the distance of the target) was shown in [11, 27, 4], culminating in a hardness factor of $2^{\log^{1-\varepsilon} n}$ for any $\varepsilon > 0$ [17] under the assumption that NP is not in randomized quasi-polynomial time. On the positive side, polynomial time algorithms for the approximate search version of CVPP were studied (implicitly) in [5, 18], where the current best approximation factor $O(n/\sqrt{\log n})$ was recently achieved in [7]. For the gap decisional version of CVPP, the results are better, where the current best approximation factor is $O(\sqrt{n/\log n})$ [1].

[2] The SVP and CVP can be defined over any norm, though we restrict our attention here to the Euclidean norm.

Exact CVPP **algorithms.** Given the hardness results for polynomial sized preprocessing, we do not expect efficient algorithms for solving exact CVPP for general lattices. For applications in wireless coding however, one has control over the coding lattice, though constructing coding lattices with good error correcting properties (i.e. large minimum distance) for which decoding is "easy" remains an outstanding open problem. In this context, the study of fast algorithms for exact CVPP in general lattices can yield new tools in the context of lattice design, as well as new insights for solving CVP without preprocessing.

The extant algorithms for exact CVPP are in fact also algorithms for CVP, that is, the time to compute the preprocessing is bounded by query / search time. There are currently two classes of CVP algorithms which fit the preprocessing / search model (this excludes only the randomized sieving approaches [2, 3]).

The first class is based on lattice basis reduction [20], which use a "short" lattice basis as preprocessing to solve lattice problems, that is polynomial sized preprocessing. The fastest such algorithm is due to Kannan [16], with subsequent refinements in [15, 5, 14, 26], which computes a Hermite-Korkine-Zolatoreff basis (HKZ) during the preprocessing phase in $\widetilde{O}(n^{\frac{n}{2e}})$[3] time and $\text{poly}(n)$ space, and in the query phase uses a search tree to compute the coefficients of the closest vector under the HKZ basis in $\widetilde{O}(n^{\frac{n}{2}})$ time and $\text{poly}(n)$ space.

The second class, which are the most relevant to this work, use the Voronoi cell (see Section 4.1.1 for precise definitions) of the lattice – the centrally symmetric polytope corresponding to the points closer to the origin than to other lattice points – as preprocessing, and were first introduced by Sommer, Feder and Shalvi [28]. In [28], they give an iterative procedure that uses the facet inducing lattice vectors of the Voronoi cell (known as the *Voronoi relevant vectors*) to move closer and closer to the target, and show that this procedure converges to a closest lattice vector in a finite number of steps. The number of Voronoi relevant vectors is $2(2^n - 1)$ in the worst-case (this holds for almost all lattices), and hence Voronoi cell based algorithms often require exponential size preprocessing. Subsequently, Micciancio and Voulgaris [24] (henceforth MV), showed how to compute the Voronoi relevant vectors during preprocessing and how to implement the search phase such that each phase uses $\widetilde{O}(4^n)$ time and $\widetilde{O}(2^n)$ space (yielding the first $2^{O(n)}$ time algorithm for exact CVP!).

While Voronoi cell based CVPP algorithms require exponential time and space on general lattices, it was recently shown in [23] that a variant of [28] can be implemented in polynomial time for lattices of Voronoi's first kind – lattices which admit a set of $n + 1$ generators whose Gram matrix is the Laplacian of a non-negatively weighted graph – using these generators as the preprocessing advice. Hence, it is sometimes possible to "scale down" the complexity of exact solvers for interesting classes of lattices.

Main Result. Our main result is a randomized $\widetilde{O}(2^n)$ expected time and space algorithm for exact CVPP, improving the $\widetilde{O}(4^n)$ (deterministic) running time of MV. Our preprocessing is the same as MV, that is we use the facet inducing lattice vectors of the Voronoi cell, known as the *Voronoi relevant* vectors (see Figure 1), as the preprocessing advice, which in the worst case consists of $2(2^n - 1)$ lattice vectors. Our main contribution, is a new search algorithm that requires only an expected polynomial number of iterations over the set of Voronoi relevant vectors to converge to a closest lattice vector, compared to $\widetilde{O}(2^n)$ in MV.

One minor caveat to our iteration bound is that unlike that of MV, which only depends on n, ours also depends (at worst linearly) on the binary encoding length of the input lattice basis and target (though the $\widetilde{O}(2^n)$ bound also holds for our procedure). Hence, while the bound is polynomial, it is only "weakly" so. In applications however, it is rather anomalous to encounter n dimensional lattice bases and targets whose individual coefficients require more than say $\text{poly}(n)$ bits to represent, and hence the iteration bound will be $\text{poly}(n)$ in almost all settings of relevance. Furthermore, it is unclear if this dependence of our algorithm is inherent, or whether it is just an artifact of the analysis.

[3] The $\widetilde{O}$ notation suppresses polylogarithmic factors.

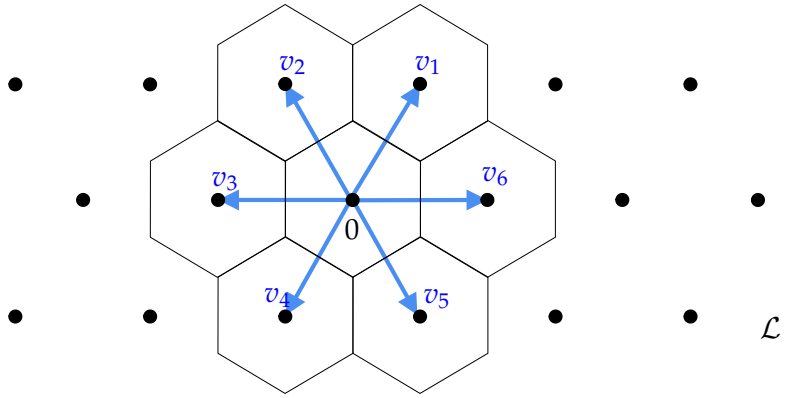


Figure 1: The Voronoi cell and its relevant vectors

While our algorithm is randomized, it is Las Vegas, and hence the randomness is in the runtime and not the correctness. Furthermore, the amount of randomness we require is polynomial: it corresponds to the randomness needed to generate a nearly-uniform sample from the Voronoi cell, which can be achieved using Monte Carlo Markov Chain (MCMC) methods over convex bodies [9, 22]. This requires a polynomial number of calls to a membership oracle. Each membership oracle test requires an enumeration over the $\widetilde{O}(2^n)$ Voronoi-relevant vectors, resulting in a total complexity of $\widetilde{O}(2^n)$.

Unfortunately, we do not know how to convert our CVPP improvement to one for CVP. The technical difficulty lies in the fact that computing the Voronoi relevant vectors, using the current approach, is reduced to solving $\widetilde{O}(2^n)$ related lower dimensional CVPs on an $n-1$ dimensional lattice (for which the Voronoi cell has already been computed). While the MV CVPP algorithm requires $\widetilde{O}(4^n)$ for worst case targets (which we improve to $\widetilde{O}(2^n)$), they are able to use the relations between the preprocessing CVPs to solve each of them in amortized $\widetilde{O}(2^n)$ time per instance. Hence, with the current approach, reducing the running time of CVP to $\widetilde{O}(2^n)$ would require reducing the amortized per instance complexity to polynomial, which seems very challenging.

Organization. In the next section, section 2, we explain how to solve CVPP by finding short paths over the Voronoi graph. In particular, we review the iterative slicer [28] and MV [24] algorithms for navigating the Voronoi graph, and describe our new *randomized straight line procedure* for this task. In section 3, we state the guarantees for the randomized straight line procedure and use it to give our expected $\widetilde{O}(2^n)$ time CVPP algorithm (Theorem 8), as well as an optimal relationship between geometric and path distance on the Voronoi graph (Theorem 5). The main geometric estimates underlying the analysis of the randomized straight path algorithm are proved in section 5.

Definitions and references for the concepts and prior algorithms used in the paper can be found in section 4. In particular, see subsections 4.1 for basic lattice definitions, and subsection 4.1.1 for precise definitions and fundamental facts about the Voronoi cell and related concepts.

2 Navigating the Voronoi graph

In this section, we explain how one can solve CVP using an efficient navigation algorithm over the Voronoi graph of a lattice. We first describe the techniques used by [28, 24] for finding short paths on this graph, and then give our new (randomized) approach.

Paths on the Voronoi graph. Following the strategy of [28, 24], our search algorithm works on the Voronoi graph $\mathcal{G}$ of an n dimensional lattice $\mathcal{L}$.

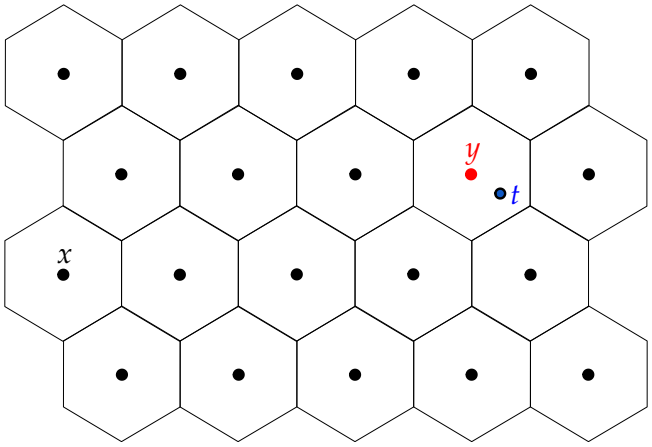


Figure 2: CVP solution is the center of target-containing Voronoi cell

Definition 1 (Voronoi Cell). *The Voronoi cell $\mathcal{V}(\mathcal{L})$ of $\mathcal{L}$ is defined as*

$$\begin{aligned}\mathcal{V}(\mathcal{L}) &= \{\mathbf{x} \in \mathbb{R}^n : \|\mathbf{x}\|_2 \leq \|\mathbf{x}-\mathbf{y}\|_2,\ \forall \mathbf{y} \in \mathcal{L} \setminus \{\mathbf{0}\}\} \\ &= \{\mathbf{x} \in \mathbb{R}^n : \langle \mathbf{x}, \mathbf{y}\rangle \leq \langle \mathbf{y}, \mathbf{y}\rangle /2,\ \forall \mathbf{y} \in \mathcal{L} \setminus \{\mathbf{0}\}\},\end{aligned}$$

the set of points closer to $\mathbf{0}$ than any other lattice point. When the lattice in question is clear, we simply write $\mathcal{V}$ for $\mathcal{V}(\mathcal{L})$. It was shown by Voronoi that $\mathcal{V}$ is a centrally symmetric polytope with at most $2(2^n-1)$ facets. We define VR, *the set of* Voronoi relevant vectors *of $\mathcal{L}$, to be the lattice vectors inducing facets of $\mathcal{V}$.*

The Voronoi graph $\mathcal{G}$ is the contact graph induced by the tiling of space by Voronoi cells, that is, two lattice vectors $\mathbf{x}, \mathbf{y} \in \mathcal{L}$ are adjacent if their associated Voronoi cells $\mathbf{x}+\mathcal{V}$ and $\mathbf{y}+\mathcal{V}$ touch in a shared facet (equivalently $\mathbf{x}-\mathbf{y} \in$ VR). We denote the shortest path distance between $\mathbf{x}, \mathbf{y} \in \mathcal{L}$ on $\mathcal{G}$ by $d_{\mathcal{G}}(\mathbf{x}, \mathbf{y})$.

See Section 4.1.1 for more basic facts about the Voronoi cell.

To solve CVP on a target $\mathbf{t}$, the idea of Voronoi cell based methods is to compute a short path on the Voronoi graph $\mathcal{G}$ from a "close enough" starting vertex $\mathbf{x} \in \mathcal{L}$ to $\mathbf{t}$ (usually, a rounded version of $\mathbf{t}$ under some basis), to the center $\mathbf{y} \in \mathcal{L}$ of a Voronoi cell containing $\mathbf{t}$, which we note is a closest lattice vector by definition. (see Figure 2).

Iterative slicer. The iterative slicer [28] was the first CVP algorithm to make use of an explicit description of the Voronoi cell, in the form of the VR vectors.

The path steps of the iterative slicer are computed by greedily choosing any Voronoi relevant vector that brings the current iterate $\mathbf{z} \in \mathcal{L}$ closer to the target $\mathbf{t}$. That is, if there exists a VR vector $\mathbf{v}$ such that $\|\mathbf{z}+\mathbf{v}-\mathbf{t}\|_2 < \|\mathbf{z}-\mathbf{t}\|_2$, then we move to $\mathbf{z}+\mathbf{v}$. This procedure is iterated until there is no improving VR vector, at which point we have reached a closest lattice vector to $\mathbf{t}$. This procedure was shown to terminate in a finite number of steps, and currently, no good quantitative bound is known on its convergence time.

The Voronoi norm. We now make precise which notion of closeness to the target we use (as well as MV) for the starting lattice vector $\mathbf{x}$ to the target $\mathbf{t}$. Notice that for the path finding approach to make sense from the perspective of CVP, we need to start the process from a point $\mathbf{x} \in \mathcal{L}$ that we know is apriori close in graph distance to a closest lattice vector $\mathbf{y}$ to $\mathbf{t}$. Given the complexity of $\mathcal{G}$ and the fact that we do not know $\mathbf{y}$, we will need a robust proxy for graph distance that we can estimate knowing only $\mathbf{x}$ and $\mathbf{t}$. From this perspective, it was shown in [24] that the Voronoi norm

$$\|\mathbf{t}-\mathbf{x}\|_{\mathcal{V}} = \inf\{s \geq 0 : \mathbf{t}-\mathbf{x} \in s\mathcal{V}\} = \sup_{\mathbf{v} \in \mathrm{VR}} 2\frac{\langle \mathbf{v}, \mathbf{t}-\mathbf{x}\rangle}{\langle \mathbf{v}, \mathbf{v}\rangle}$$

of $\mathbf{t}-\mathbf{x}$ (i.e. the smallest scaling of $\mathcal{V}$ containing $\mathbf{t}-\mathbf{x}$) can be used to bound the shortest path distance between $\mathbf{x}$ and $\mathbf{y}$. Here the quantity $\|\mathbf{t}-\mathbf{x}\|_{\mathcal{V}}$ is robust in the sense that $\|\mathbf{y}-\mathbf{x}\|_{\mathcal{V}} \leq \|\mathbf{t}-\mathbf{x}\|_{\mathcal{V}} + \|\mathbf{y}-\mathbf{t}\|_{\mathcal{V}} \leq \|\mathbf{t}-\mathbf{x}\|_{\mathcal{V}} + 1$ by the triangle inequality. Hence from the perspective of the Voronoi norm, $\mathbf{t}$ is simply a "noisy" version of $\mathbf{y}$. Furthermore, given that each Voronoi relevant vector has Voronoi norm 2, one can construct a lattice vector $\mathbf{x}$ such that $\|\mathbf{t}-\mathbf{x}\|_{\mathcal{V}} \leq n$, by simply expressing $\mathbf{t} = \sum_{i=1}^{n} a_i \mathbf{v}_i$, for some linearly independent $\mathbf{v}_1, \ldots, \mathbf{v}_n \in \mathrm{VR}$, and letting $\mathbf{x} = \sum_{i=1}^{n} \lceil a_i \rfloor \mathbf{v}_i$.

The MV Path. We now present the MV path finding approach, and give the relationship they obtain between $\|\mathbf{t}-\mathbf{x}\|_{\mathcal{V}}$ and the path distance to a closest lattice vector $\mathbf{y}$ to $\mathbf{t}$.

The base principle of MV [24] is similar to that of the iterative slicer, but it uses a different strategy to select the next VR vector to follow, resulting in a provably single exponential path length.

In MV, a path step consists of tracing the straight line from the current path vertex $\mathbf{z} \in \mathcal{L}$ to the target $\mathbf{t}$, and moving to $\mathbf{z}+\mathbf{v}$ where $\mathbf{v} \in \mathrm{VR}$ induces a facet (generically unique) of $\mathbf{z}+\mathcal{V}$ crossed by the line segment $[\mathbf{z}, \mathbf{t}]$. It is not hard to check that each step can be computed using $O(n|\mathrm{VR}|) = \widetilde{O}(2^n)$ arithmetic operations, and hence the complexity of computing the path is $O(n|\mathrm{VR}| \times \text{ path length})$.

The main bound they give on the path length, is that if the start vertex $\mathbf{x} \in 2\mathcal{V}+\mathbf{t}$ (i.e. Voronoi distance less than 2), then the path length is bounded by 2^n. To prove the bound, they show that the path always stays inside $\mathbf{t}+2\mathcal{V}$, that the ℓ_2 distance to the target monotonically decreases along the path (and hence it is acyclic), and that the number of lattice vectors in the interior of $\mathbf{t}+2\mathcal{V}$ is at most 2^n.

To build the full path, they run this procedure on the Voronoi graph for decreasing exponential scalings of $\mathcal{L}$ [4], and build a path (on a supergraph of $\mathcal{G}$) of length $O(2^n \log_2 \|\mathbf{t}-\mathbf{x}\|_{\mathcal{V}})$. One can also straightforwardly adapt the MV procedure to stay on $\mathcal{G}$, by essentially breaking up the line segment $[\mathbf{x}, \mathbf{t}]$ in pieces of length at most 2, yielding a path length of $O(2^n \|\mathbf{x}-\mathbf{t}\|_{\mathcal{V}})$. Since we can always achieve a starting distance of $\|\mathbf{x}-\mathbf{t}\|_{\mathcal{V}} \leq n$ by straightforward basis rounding, note that the distance term is lower order compared to the proportionality factor 2^n.

Randomized Straight Line. Given the 2^n proportionality factor between geometric and path distance achieved by the MV algorithm, the main focus of our work will be to reduce the proportionality factor to polynomial. In fact, will show the existence of paths of length $(n/2)(\|\mathbf{t}-\mathbf{x}\|_{\mathcal{V}}+1)$, however the paths we are able to construct will be longer.

For our path finding procedure, the base idea is rather straightforward, we simply attempt to follow the sequence of Voronoi cells on the straight line from the start point $\mathbf{x}$ to the target $\mathbf{t}$. We dub this procedure the straight line algorithm. As we will show, the complexity of computing this path follows the same pattern as MV (under certain genericity assumptions), and hence the challenge is proving that the number of Voronoi cells the path crosses is polynomial. Unfortunately, we do not know how to analyze this procedure directly. In particular, we are unable to rule out the possibility that a "short" line segment (say of Voronoi length $O(1)$) may pass through exponentially many Voronoi cells in the worst case (though we do not have any examples of this).

To get around the problem of having "unexpectedly many" crossings, we will make use of randomization to perturb the starting point of the line segment. Specifically, we will use a *randomized straight line* path from $\mathbf{x} \in \mathcal{L}$ to $\mathbf{t}$ which proceeds as follows (see Figure 3):

(A) Move to $\mathbf{x}+Z$, where $Z \sim \mathrm{Uniform}(\mathcal{V})$ is sampled uniformly from the Voronoi cell.

(B) Follow the line from $\mathbf{x}+Z$ to $\mathbf{t}+Z$.

(C) Follow the line from $\mathbf{t}+Z$ to $\mathbf{t}$.

[4] The MV path is in fact built on a supergraph of the Voronoi graph, which has edges corresponding to $2^i\mathrm{VR}$, $i \geq 0$.

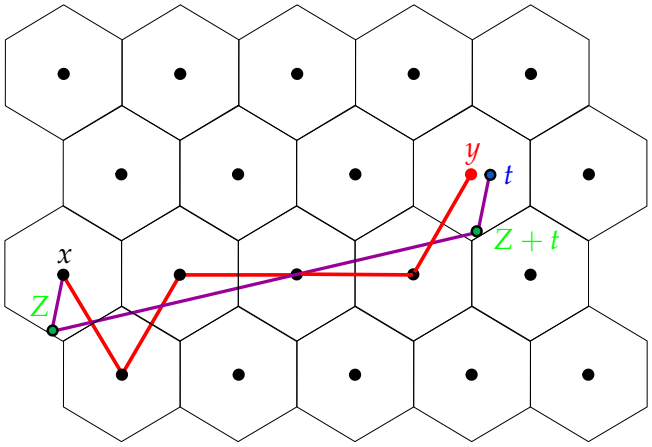


Figure 3: Randomized Straight Line algorithm

We briefly outline the analysis bounding the expected number of Voronoi cells this path crosses, which we claim achieves a polynomial proportionality factor with respect to $\|\mathbf{t} - \mathbf{x}\|_{\mathcal{V}}$.

To begin, note that in phase A, we stay entirely within $\mathbf{x} + Z$, and hence do not cross any Voronoi cells.

In phase B, at every time $\alpha \in [0, 1]$, the point $(1 - \alpha)\mathbf{x} + \alpha\mathbf{t} + Z$ is in a uniformly random coset of $\mathbb{R}^n/\mathcal{L}$ since Z is uniform. Hence the probability that we we cross a boundary between time α and $\alpha + \varepsilon$ is identical to the probability that we cross a boundary going from Z to $Z + \varepsilon(\mathbf{t} - \mathbf{x})$. Taking the limit as $\varepsilon \to 0$ and using linearity of expectation, we use the above invariance to show that the expected number of boundaries we cross is bounded by $(n/2)\|\mathbf{t} - \mathbf{x}\|_{\mathcal{V}}$, the Voronoi distance between $\mathbf{x}$ and $\mathbf{t}$. In essence, we relate the number of crossings to the probability that a uniform sample from $\mathcal{V}$ (equivalently, a uniform coset) is close under the Voronoi norm to the boundary $\partial\mathcal{V}$, which is a certain surface area to volume ratio.

Interestingly, as a consequence of our bound for phase B, we are able to give an optimal relationship between the Voronoi distance between two lattice points and their shortest path distance on $\mathcal{G}$, which we believe to be independent interest. In particular, for two lattice points $\mathbf{x}, \mathbf{y} \in \mathcal{L}$, we show in Theorem 5 that the shortest path distance on $\mathcal{G}$ is at least $\|\mathbf{x} - \mathbf{y}\|_{\mathcal{V}}/2$ and at most $(n/2)\|\mathbf{x} - \mathbf{y}\|_{\mathcal{V}}$, which is tight for certain pairs of lattice points on $\mathbb{Z}^n$.

It remains now to bound the expected number of crossings in phase C. Here, the analysis is more difficult than the second step, because the random shift is only on one side of the line segment from $\mathbf{t} + Z$ to $\mathbf{t}$. We will still be able to relate the expected number of crossings to "generalized" surface area to volume ratios, however the probability distributions at each time step will no longer be invariant modulo the lattice. In particular, the distributions become more concentrated as we move closer to $\mathbf{t}$, and hence we slowly lose the benefits of the randomness as we get closer to $\mathbf{t}$. Unfortunately, because of this phenomenon, we are unable to show in general that the number of crossings from $\mathbf{t} + Z$ to $\mathbf{t}$ is polynomial. However, we will be able to bound the number of crossings from $\mathbf{t} + Z$ to $\mathbf{t} + \alpha Z$ by $O(n \ln(1/\alpha))$, that is, a very slow growing function of α as $\alpha \to 0$. Fortunately, for rational lattices and targets, we can show that for α not too small, in particular $\ln(1/\alpha)$ linear in the size of binary encoding of the basis and target suffices, that $\mathbf{t} + \alpha Z$ and $\mathbf{t}$ lie in the same Voronoi cell. This yields the claimed (weakly) polynomial bound.

3 Analysis and Applications of Randomized Straight Line

In this section, we give the formal guarantees for the randomized straight line algorithm and its applications. The analysis here will rely on geometric estimates for the number of crossings, whose

proofs are found in Section 5.

To begin, we make formal the connection between Voronoi cells crossings, the length of the randomized straight line path, and the complexity of computing it.

Lemma 2 (Randomized Straight Line Complexity)**.** *Let $\mathbf{x} \in \mathcal{L}$ be the starting point and let $\mathbf{t} \in \mathbb{R}^n$ be the target. Then using perturbation $Z \sim \text{Uniform}(\mathcal{V})$, the expected edge length of the path from $\mathbf{x}$ to a closest lattice vector $\mathbf{y}$ to $\mathbf{t}$ on $\mathcal{G}$ induced by the randomized straight line procedure is*

$$\mathbb{E}[|(\mathcal{L} + \partial\mathcal{V}) \cap [\mathbf{x} + Z, \mathbf{t} + Z]|] + \mathbb{E}[|(\mathcal{L} + \partial\mathcal{V}) \cap [\mathbf{t} + Z, \mathbf{t})|] \, .$$

Furthermore, with probability 1, each edge of the path can be computed using $O(n|\text{VR}|)$ arithmetic operations.

While rather intuitive, the proof of this Lemma is somewhat tedious, and so we defer it to section 6. Note that $(\mathcal{L} + \partial\mathcal{V}) \cap [\mathbf{x} + Z, \mathbf{t} + Z]$ corresponds to the phase B crossings, and that $(\mathcal{L} + \partial\mathcal{V}) \cap [\mathbf{t} + Z, \mathbf{t})$ corresponds to the phase C crossings.

Our bound for the phase B crossings, which is proved in Section 5.1, is as follows.

Theorem 3 (Phase B crossing bound)**.** *Let $\mathcal{L}$ be an n dimensional lattice. Then for $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$ and $Z \sim$ uniform$(\mathcal{V})$, we have that*

$$\mathbb{E}_Z[|(\mathcal{L} + \partial\mathcal{V}) \cap [\mathbf{x} + Z, \mathbf{y} + Z]|] \leq (n/2)\|\mathbf{y} - \mathbf{x}\|_{\mathcal{V}} \, .$$

For phase C, we give a bound on the number crossings for a truncation of the phase C path. That is, instead of going all the way from $\mathbf{t} + Z$ to $\mathbf{t}$, we stop at $\mathbf{t} + \alpha Z$, for $\alpha \in (0, 1]$. Its proof is given in Section 5.2.

Theorem 4 (Phase C crossing bound)**.** *For $\alpha \in (0, 1]$, $Z \sim \text{Uniform}(\mathcal{V})$, $n \geq 2$, we have that*

$$\mathbb{E}[|(\mathcal{L} + \partial\mathcal{V}) \cap [Z + \mathbf{t}, \alpha Z + \mathbf{t}]|] \leq \frac{e^2}{\sqrt{2} - 1} n(2 + \ln(4/\alpha)) \, .$$

Using the crossing estimate for phase B, we now show that from the perspective of existence, one can improve the MV proportionality factor between geometric and path distance from exponential to linear in dimension.

Theorem 5. *For $\mathbf{x}, \mathbf{y} \in \mathcal{L}$, we have that*

$$(1/2)\|\mathbf{x} - \mathbf{y}\|_{\mathcal{V}} \leq d_{\mathcal{G}}(\mathbf{x}, \mathbf{y}) \leq (n/2)\|\mathbf{x} - \mathbf{y}\|_{\mathcal{V}}.$$

Furthermore, the above is best possible, even when restricted to $\mathcal{L} = \mathbb{Z}^n$.

Proof. For the lower bound, note that $d_{\mathcal{G}}(\mathbf{x}, \mathbf{y})$ is the minimum $k \in \mathbb{Z}_+$ such that there exists $\mathbf{v}_1, \ldots, \mathbf{v}_k \in$ VR satisfying $\mathbf{y} = \mathbf{x} + \sum_{i=1}^k \mathbf{v}_i$. Since $\forall \mathbf{v} \in$ VR, $\|\mathbf{v}\|_{\mathcal{V}} = 2$, by the triangle inequality

$$\|\mathbf{y} - \mathbf{x}\|_{\mathcal{V}} = \|\sum_{i=1}^{k} \mathbf{v}_i\|_{\mathcal{V}} \leq \sum_{i=1}^{k} \|\mathbf{v}_i\|_{\mathcal{V}} = 2k,$$

as needed.

For the upper bound, we run the randomized straight line procedure from $\mathbf{x}$ to $\mathbf{y}$, i.e. setting $\mathbf{t} = \mathbf{y}$. By Lemma 2, the expected path length on $\mathcal{G}$ is

$$\mathbb{E}[|(\mathcal{L} + \partial\mathcal{V}) \cap [\mathbf{x} + Z, \mathbf{y} + Z]|] + \mathbb{E}[|(\mathcal{L} + \partial\mathcal{V}) \cap [\mathbf{y} + Z, \mathbf{y})|]$$

where $Z \sim \text{Uniform}(\mathcal{V})$. Since $\mathbf{y} \in \mathcal{L}$ and $Z \in \text{int}(\mathcal{V})$ with probability 1, note that

$$\mathbb{E}[|(\mathcal{L} + \partial\mathcal{V}) \cap [\mathbf{y} + Z, \mathbf{y})|] = 0\,,$$

i.e. the number of steps in phase C is 0. It therefore suffices to bound the number phase B steps. By Theorem 3, we have that

$$\mathbb{E}[|(\mathcal{L} + \partial\mathcal{V}) \cap [\mathbf{x} + Z, \mathbf{y} + Z]|] \leq (n/2)\|\mathbf{x} - \mathbf{y}\|_{\mathcal{V}}\,,$$

as needed. This shows the desired upper bound on the path length.

We now show that the above bounds are sharp. For the lower bound, note that it is tight for any two adjacent lattice vectors, since $\forall \mathbf{v} \in \text{VR}$, $\|\mathbf{v}\|_{\mathcal{V}} = 2$. For the upper bound, letting $\mathcal{L} = \mathbb{Z}^n$, $\mathcal{V} = [-1/2, 1/2]^n$, $\text{VR} = \{\pm\mathbf{e}_1, \ldots, \pm\mathbf{e}_n\}$, the shortest path between $\mathbf{x} = \mathbf{0}$ and $\mathbf{y} = (1, \ldots, 1)$ has length n, while $\|\mathbf{x} - \mathbf{y}\|_{\mathcal{V}} = 2\|\mathbf{x} - \mathbf{y}\|_\infty = 2$. □

Since the Voronoi distance changes by at most 1 when switching from $\mathbf{y}$ to $\mathbf{t} \in \mathbf{y} + \mathcal{V}$, we note that the above bound immediately yields a corresponding bound on the path length to a closest lattice vector to any target.

As the phase C bound in Theorem 4 only holds for the truncated path, it does yield a bound on the randomized straight line path length for general lattices. However, for rational lattices and targets, we now show that for α small enough, the truncated path in fact suffices.

We will derive this result from the following simple Lemmas.

Lemma 6 (Rational Lattice Bound)**.** *Let $\mathcal{L} \subseteq \mathbb{Q}^n$, and $\mathbf{t} \in \mathbb{Q}^n$. Let $\bar{q} \in \mathbb{N}$ be the smallest number such that $\bar{q}\mathcal{L} \subseteq \mathbb{Z}^n$ and $\bar{q}\mathbf{t} \in \mathbb{Z}^n$, and let $\mu = \mu(\mathcal{L})$ denote the covering radius of $\mathcal{L}$. For $\mathbf{y} \in \mathcal{L}$, if $\mathbf{t} \notin \mathbf{y} + \mathcal{V}$, then*

$$\|\mathbf{t} - \mathbf{y}\|_{\mathcal{V}} \geq 1 + 1/(2\bar{q}\mu)^2\,.$$

Proof. Note that $\mathbf{t} \notin \mathbf{y} + \mathcal{V}$ iff $\|\mathbf{t} - \mathbf{y}\|_{\mathcal{V}} > 1$. From here, we have that

$$1 < \|\mathbf{t} - \mathbf{y}\|_{\mathcal{V}} = 2\frac{\langle \mathbf{v}, \mathbf{t} - \mathbf{y}\rangle}{\langle \mathbf{v}, \mathbf{v}\rangle},$$

for some $\mathbf{v} \in \text{VR}$. By our assumptions, we note that $\langle \mathbf{v}, \mathbf{t} - \mathbf{y}\rangle = a/q^2$, for $a \in \mathbb{N}$. Next, $\|\mathbf{v}\|_2 \leq 2\mu$ (see the end of section 4.1.1 for details) and $\mathbf{v} \in \mathbb{Z}^n/q$, and hence we can write $\langle \mathbf{v}, \mathbf{v}\rangle = b/q^2$, $b \in \mathbb{N}$, for $b \leq (2\bar{q}\mu)^2$. Therefore $1 < \|\mathbf{t} - \mathbf{y}\|_{\mathcal{V}} = \frac{2a}{b}$ implies that $2a > b$. Since $a, b \in \mathbb{N}$, we must have that

$$\|\mathbf{t} - \mathbf{y}\|_{\mathcal{V}} = \frac{2a}{b} \geq \frac{b+1}{b} = 1 + \frac{1}{b} \geq 1 + \frac{1}{(2\bar{q}\mu)^2}$$

as needed. □

The following shows that the relevant quantities in Lemma 6 can be bounded by the binary encoding length of the lattice basis and target. Since it is rather standard, we defer the proof to section 6.

Lemma 7 (Bit Length Bound)**.** *Let $B \in \mathbb{Q}^{n\times n}$ be a lattice basis matrix for an n dimensional lattice $\mathcal{L}$, with $B_{ij} = \frac{p_{ij}^B}{q_{ij}^B}$ where $p_{ij}^B \in \mathbb{Z}$ and $q_{ij}^B \in \mathbb{N}$. Let $\mathbf{t} \in \mathbb{Q}^n$, with $\mathbf{t}_i = \frac{p_i^{\mathbf{t}}}{q_i^{\mathbf{t}}}$, $p_i^{\mathbf{t}} \in \mathbb{Z}$, $q_i^{\mathbf{t}} \in \mathbb{N}$. Then for $\bar{q} \in \mathbb{N}$, the smallest number such that $\bar{q}\mathcal{L} \subseteq \mathbb{Z}^n$ and $\bar{q}\mathbf{t} \in \mathbb{Z}^n$, we have that $\log_2(\bar{q}\mu(\mathcal{L})) \leq \text{enc}(B) + \text{enc}(\mathbf{t})$ and $\log_2(\mu(\mathcal{L})/\lambda_1(\mathcal{L})) \leq \text{enc}(B)$.*

We are now in a position to give our full CVPP algorithm.

Theorem 8 (CVPP Algorithm). *Let $\mathcal{L}$ be an n-dimensional lattice with basis $B \in \mathbb{Q}^{n\times n}$, let* VR *denote the set of Voronoi relevant vectors of $\mathcal{L}$. Given the set* VR *as preprocessing, for any target $\mathbf{t} \in \mathbb{Q}^n$, a closest lattice vector to $\mathbf{t}$ can be computed using an expected* $\mathrm{poly}(n, \mathrm{enc}(B), \mathrm{enc}(t))|\mathrm{VR}|$ *arithmetic operations.*

Proof. To start we pick linearly independent $\mathbf{v}_1, \ldots, \mathbf{v}_n \in \mathrm{VR}$. We then compute the coefficent representation of $\mathbf{t}$ with respect to $\mathbf{v}_1, \ldots, \mathbf{v}_n$, that is $\mathbf{t} = \sum_{i=1}^n a_i \mathbf{v}_i$. From here we compute the lattice vector $\mathbf{x} = \sum_{i=1}^n \lceil a_i \rfloor \, \mathbf{v}_i$, i.e. the rounding of $\mathbf{t}$.

Next, using the convex body sampler (Theorem 18), we compute a $(1/4)$-uniform sample Z over $\mathcal{V}$. Note that a membership oracle for $\mathcal{V}$ can be implemented using $O(n|\mathrm{VR}|)$ arithmetic operations. Furthermore, letting $\lambda_1 = \lambda_1(\mathcal{L})$, $\mu = \mu(\mathcal{L})$, we have that

$$(\lambda_1/2)B_2^n \subseteq \mathcal{V} \subseteq \mu B_2^n \,,$$

where $\lambda_1 = \min_{\mathbf{v}\in\mathrm{VR}} \|\mathbf{v}\|_2$, $(1/2)\max_{\mathbf{v}\in\mathrm{VR}} \|\mathbf{v}\|_2 \leq \mu \leq (\sqrt{n}/2)\max_{\mathbf{v}\in\mathrm{VR}} \|\mathbf{v}\|_2$ (see Lemma 15 in the Appendix). Hence, nearly tight sandwiching estimates for $\mathcal{V}$ can be easily computed using the set VR.

We now run the randomized straight line algorithm starting at lattice point $\mathbf{x}$, perturbation Z, and target $\mathbf{t}$. If the path produced by the algorithm becomes longer than $cn(n + (\mathrm{enc}(B) + \mathrm{enc}(\mathbf{t})))$ (for some $c \geq 1$ large enough), restart the algorithm, and otherwise return the found closest lattice vector.

The correctness of the algorithm follows directly from the correctness of the randomized straight line algorithm (Lemma 2), and hence we need only show a bound on the expected runtime.

Runtime. We first bound the number of operations performed in a single iteration. Computing $\mathbf{v}_1, \ldots, \mathbf{v}_n$, $\mathbf{t}$, and the sandwiching estimates for $\mathcal{V}$, requires at most $O(n^3|\mathrm{VR}|)$ arithmetic operations. By Lemma 7, the convex body sampler requires at most

$$\mathrm{poly}(n, \log(\sqrt{n}\mu/\lambda_1))|\mathrm{VR}| = \mathrm{poly}(n, \mathrm{enc}(B))|\mathrm{VR}|$$

arithmetic operations. For the randomized straight line algorithm, each step requires at most $O(n|\mathrm{VR}|)$ arithmetic operations by Lemma 2. Since we truncate it at $O(n(n + (\mathrm{enc}(B) + \mathrm{enc}(\mathbf{t}))))$ iterations, this requires at most $O(n^2(n + (\mathrm{enc}(B) + \mathrm{enc}(\mathbf{t})))|\mathrm{VR}|)$ arithmetic operations. Hence the total number of arithmetic operations per iteration is bounded by

$$\mathrm{poly}(n, \mathrm{enc}(B), \mathrm{enc}(t))|\mathrm{VR}|.$$

We now show that the algorithm performs at most $O(1)$ iterations on expectation. For this it suffices to show that each iteration succeeds with constant probability. In particular, we will show that with constant probability, the length of the randomized straight line path is bounded by $O(n^2(\mathrm{enc}(B) + \mathrm{enc}(\mathbf{t})))$. To do this we will simply show that the expected path length is bounded by $O(n^2(\mathrm{enc}(B) + \mathrm{enc}(\mathbf{t})))$ under the assumption that Z is truly uniform. By Markov's inequality, the probability that the length is less than twice the expectation is at least $1/2$ for a truly uniform Z, and hence it will be a least $1/4$ for a $1/4$-uniform Z.

To begin, we note that by the triangle inequality

$$\|\mathbf{t} - \mathbf{x}\|_{\mathcal{V}} \leq \sum_{i=1}^n |a_i - \lceil a_i \rfloor| \|\mathbf{v}_i\|_{\mathcal{V}} \leq \sum_{i=1}^n (1/2)(2) = n \,.$$

Let $\bar{q}$ be as in Lemma 7, and let $\alpha = \frac{1}{(4\bar{q}\mu)^2}$, where we have that $\ln(1/\alpha) = O(\mathrm{enc}(B) + \mathrm{enc}(\mathbf{t}))$. Let $\mathbf{y} \in \mathcal{L}$ denote the center of the first Voronoi cell containing $\mathbf{t} + \alpha Z$ found by the randomized straight line algorithm. We claim that $\mathbf{y}$ is a closest lattice vector to $\mathbf{t}$, or equivalently that $\mathbf{t} \in \mathbf{y} + \mathcal{V}$. Assume not, then by Lemma 6, $\|\mathbf{t} - \mathbf{y}\|_{\mathcal{V}} \geq 1 + \frac{1}{(2\bar{q}\mu)^2}$. On the other hand, since $\mathbf{t} + \alpha Z \in \mathbf{y} + \mathcal{V}$ and $Z \in \mathcal{V}$, by the triangle inequality

$$\|\mathbf{t} - \mathbf{y}\|_{\mathcal{V}} \leq \|\mathbf{t} - (\mathbf{t} + \alpha Z)\|_{\mathcal{V}} + \|(\mathbf{t} + \alpha Z) - \mathbf{y}\|_{\mathcal{V}} \leq \alpha + 1 = 1 + \frac{1}{(4\bar{q}\mu)^2} \,,$$

a contradiction. Hence $\mathbf{y}$ is a closest lattice vector to $\mathbf{t}$. If $Z \sim \text{Uniform}(\mathcal{V})$, then by Theorems 3 and 4 the expected length of the randomized straight line path up till $\mathbf{t} + \alpha Z$ (i.e. till we find $\mathbf{y}$) is bounded by

$$\begin{aligned}(n/2)\|\mathbf{t}-\mathbf{x}\|_{\mathcal{V}} + \frac{e^2}{\sqrt{2}-1} n(2+\ln(4/\alpha)) &= n^2/2 + \frac{e^2}{\sqrt{2}-1} n(2+2\ln(8\bar{q}\mu)) \\ &= O(n(n+(\text{enc}\,(B)+\text{enc}\,(t)))),\end{aligned}$$

as needed. The theorem thus follows. □

4 Preliminaries

Basics. For $n \geq 1$, we denote $\mathbb{R}^n, \mathbb{Q}^n, \mathbb{Z}^n$ to be the set of n dimensional real / rational / integral vectors respectively. We let $\mathbb{N}$ denote the set of natural numbers, and $\mathbb{Z}_+$ denote the set of non-negative integers. For two sets $A, B \subseteq \mathbb{R}^n$, we denote their Minkowski sum $A + B = \{\mathbf{a} + \mathbf{b} : \mathbf{a} \in A, \mathbf{b} \in B\}$. We write ∂A to denote the topological boundary of A. For a set $A \subseteq \mathbb{R}^n$, its affine hull, affhull(A), is the inclusion wise smallest linear affine space containing A. We denote the interior of A in $\mathbb{R}^n$ as int(A). We denote the relative interior of A by relint(A), which is the interior of A with the respect to the subspace topology on affhull(A).

For two n dimensional vectors $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$, we denote their inner product $\langle \mathbf{x}, \mathbf{y} \rangle = \sum_{i=1}^n \mathbf{x}_i \mathbf{y}_i$. The ℓ_2 (Euclidean) norm of a vector $\mathbf{x}$ is denoted $\|\mathbf{x}\|_2 = \sqrt{\langle \mathbf{x}, \mathbf{x} \rangle}$. We let $B_2^n = \{\mathbf{x} \in \mathbb{R}^n : \|\mathbf{x}\|_2 \leq 1\}$ denote the unit Euclidean ball, and let $S^{n-1} = \partial B_2^n$ denote the unit sphere. For vectors $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$, we denote the closed line segment from $\mathbf{x}$ to $\mathbf{y}$ by $[\mathbf{x}, \mathbf{y}] \stackrel{\text{def}}{=} \{\alpha \mathbf{x} + (1-\alpha)\mathbf{y} : \alpha \in [0,1]\}$, and $[\mathbf{x}, \mathbf{y})$ the half open line segment not containing $\mathbf{y}$.

We denote $\mathbf{e}_1, \ldots, \mathbf{e}_n$ the vectors of the standard basis of $\mathbb{R}^n$, that is the vectors such that $\mathbf{e}_i$ has a 1 in the i^{th} coordinate and 0's elsewhere.

Binary encoding. For an integer $z \in \mathbb{Z}$, the standard binary encoding for z requires $1 + \lceil \log_2(|z|+1) \rceil$ bits, which we denote enc (z). For a rational number $\frac{p}{q} \in \mathbb{Q}$, $p \in \mathbb{Z}$, $q \in \mathbb{N}$, the encoding size of $\frac{p}{q}$ is enc $\left(\frac{p}{q}\right) =$ enc $(p) +$ enc (q). For an $n \times m$ matrix $M \in \mathbb{Q}^{m \times n}$ or vector $\mathbf{a} \in \mathbb{Q}^n$, enc (M), enc $(\mathbf{a})$ denotes the sum of encoding lengths of all the entries.

Integration. We denote the k-dimensional Lebesgue measure in $\mathbb{R}^n$ by $\text{vol}_k(\cdot)$. Only $k = n$ and $k = n-1$ will be used in this paper. For $k = n-1$, we will only apply it to sets which can be written as a disjoint countable union of $n-1$ dimensional flat pieces. When integrating a function $f : \mathbb{R}^n \to \mathbb{R}$ over a set $A \subseteq \mathbb{R}^n$ using the n dimensional Lebesgue measure, we use the notation $\int_A f(\mathbf{x}) \mathrm{d}\mathbf{x}$. When integrating with respect to the $n-1$ dimensional Lebesgue measure in $\mathbb{R}^n$, we write $\int_A f(\mathbf{x}) \mathrm{dvol}_{n-1}(\mathbf{x})$.

Probability. For a random variable $X \in \mathbb{R}$, we define its expectation by $\mathbb{E}[X]$ and its variance by $\mathbb{VAR}[X] = \mathbb{E}[X^2] - \mathbb{E}[X]^2$. For two random variables $X, Y \in \Omega$, we define their total variation distance to be

$$d_{TV}(X,Y) = \max_{A \subseteq \Omega} |\Pr[X \in A] - \Pr[Y \in A]| .$$

Definition 9 (Uniform Distribution)**.** *For a set $A \subseteq \mathbb{R}^n$, we define the uniform distribution on A, denoted* Uniform(A)*, to have probability density function* $1/\text{vol}_n(A)$ *and* 0 *elsewhere. That is, for a uniform random variable $X \sim$ Uniform(A), we have that*

$$\Pr[X \in B] = \text{vol}_n(A \cap B)/\text{vol}_n(A)$$

for any measurable set $B \subseteq \mathbb{R}^n$.

Complexity. We use the notation $\widetilde{O}(T(n))$ to mean $O(T(n)\text{polylog}(T(n)))$.

4.1 Lattices

An n dimensional lattice $\mathcal{L} \subseteq \mathbb{R}^n$ is a discrete subgroup of $\mathbb{R}^n$ whose linear span is $\mathbb{R}^n$. Equivalently, $\mathcal{L}$ is generated by all integer combinations of some basis $B = (\mathbf{b}_1, \ldots, \mathbf{b}_n)$ of $\mathbb{R}^n$, i.e. $\mathcal{L} = B\mathbb{Z}^n$. For $k \in \mathbb{N}$, we define the quotient group $\mathcal{L}/k\mathcal{L} = \{\mathbf{y} + k\mathcal{L} : \mathbf{y} \in \mathcal{L}\}$. It is easy to check that the map $\mathbf{a} \to B\mathbf{a} + k\mathcal{L}$ from $(\mathbb{Z}/k\mathbb{Z})^n \stackrel{\text{def}}{=} \mathbb{Z}_k^n$ to $\mathcal{L}/(k\mathcal{L})$ is an isomorphism. In particular $|\mathcal{L}/(k\mathcal{L})| = k^n$.

A shift $\mathcal{L} + \mathbf{t}$ of $\mathcal{L}$ is called a coset of $\mathcal{L}$. The set of cosets of $\mathcal{L}$ form a group $\mathbb{R}^n/\mathcal{L}$ under addition, i.e. the torus. We will use the notation $A \pmod{\mathcal{L}}$, for a set $A \subseteq \mathbb{R}^n$, to denote the set of cosets $\mathcal{L} + A$. Note that $\mathbb{R}^n/\mathcal{L}$ is isomorphic to $[0,1)^n$ under addition $\pmod 1$ (coordinate wise), under the map $\mathbf{x} \to B\mathbf{x} + \mathcal{L}$ for any basis B of $\mathcal{L}$. We will need to make use of the uniform distribution over $\mathbb{R}^n/\mathcal{L}$, which we denote $\text{Uniform}(\mathbb{R}^n/\mathcal{L})$. To obtain a sample from $\text{Uniform}(\mathbb{R}^n/\mathcal{L})$, one can take $U \sim \text{Uniform}([0,1)^n)$ and return $BU \pmod{\mathcal{L}}$.

We denote the length of the shortest non-zero vector (or minimum distance) of $\mathcal{L}$ as $\lambda_1(\mathcal{L}) = \min_{\mathbf{y} \in \mathcal{L} \setminus \{\mathbf{0}\}} \|\mathbf{y}\|_2$. We denote the covering radius of $\mathcal{L}$ as $\mu(\mathcal{L}) = \max_{\mathbf{t} \in \mathbb{R}^n} \min_{\mathbf{y} \in \mathcal{L}} \|\mathbf{t} - \mathbf{y}\|_2$ to be the farthest distance between any point in space and the lattice.

The following standard lemma (see for instance [5]) allows us to bound the covering radius:

Lemma 10. *Let $\mathcal{L}$ be an n-dimensional lattice. If $\mathbf{v}_1, \ldots, \mathbf{v}_n \in \mathcal{L}$ are linearly independent lattice vectors, then $\mu(\mathcal{L}) \leq \frac{1}{2}\sqrt{\sum_{i=1}^n \|\mathbf{v}_i\|^2}$.*

4.1.1 Voronoi cell, tiling, and relevant vectors

For a point $\mathbf{t} \in \mathbb{R}^n$, let $\mathsf{CVP}(\mathcal{L}, \mathbf{t}) = \arg\min_{\mathbf{x} \in \mathcal{L}} \|\mathbf{t} - \mathbf{x}\|_2$, denote the set of closest lattice vectors to $\mathbf{t}$. For $\mathbf{y} \in \mathcal{L}$, let

$$H_{\mathbf{y}} = \{\mathbf{x} \in \mathbb{R}^n : \|\mathbf{x}\|_2 \leq \|\mathbf{x} - \mathbf{y}\|_2\} = \left\{\mathbf{x} \in \mathbb{R}^n : \langle \mathbf{y}, \mathbf{x} \rangle \leq \frac{1}{2}\langle \mathbf{y}, \mathbf{y} \rangle\right\},$$

denote the halfspace defining the set of points closer to $\mathbf{0}$ than to $\mathbf{y}$.

Definition 11 (Voronoi Cell)**.** *The Voronoi cell $\mathcal{V}(\mathcal{L})$ of $\mathcal{L}$ is defined as*

$$\mathcal{V}(\mathcal{L}) = \cap_{\mathbf{y} \in \mathcal{L} \setminus \{\mathbf{0}\}} H_{\mathbf{y}},$$

the set of all points in $\mathbb{R}^n$ closer or at equal distance to the origin than to any other lattice point.

Naturally, $\mathcal{V}(\mathcal{L})$ is the set of points of $\mathcal{L}$ whose closest lattice vector is $\mathbf{0}$. We abbreviate $\mathcal{V}(\mathcal{L})$ to $\mathcal{V}$ when the context is clear. It is easy to check from the definitions that a vector $\mathbf{y} \in \mathcal{L}$ is a closest lattice vector to a target $\mathbf{t} \in \mathbb{R}^n$ iff $\mathbf{t} - \mathbf{y} \in \mathcal{V}$. The CVP is then equivalent to finding a lattice shift of $\mathcal{V}$ containing the target.

From this, we see that the Voronoi cell tiles space with respect to $\mathbb{R}^n$, that is, the set of shifts $\mathcal{L} + \mathcal{V}$ cover $\mathbb{R}^n$, and shifts $\mathbf{x} + \mathcal{V}$ and $\mathbf{y} + \mathcal{V}$, $\mathbf{x}, \mathbf{y} \in \mathcal{L}$, are interior disjoint if $\mathbf{x} \neq \mathbf{y}$. From the tiling property, we have the useful property that the distribution $\text{Uniform}(\mathcal{V}) \pmod{\mathcal{L}}$ is identical to $\text{Uniform}(\mathbb{R}^n/\mathcal{L})$.

We note that the problem of separating over the Voronoi cell reduces directly to CVP, since if $\mathbf{y} \in \mathcal{L}$ is closer to a target $\mathbf{t}$ than $\mathbf{0}$, then $H_{\mathbf{y}}$ separates $\mathbf{t}$ from $\mathcal{V}$. Also, if no such closer lattice vector exists, then $\mathbf{t} \in \mathcal{V}$.

Definition 12 (Voronoi Relevant Vectors)**.** *We define $\text{VR}(\mathcal{L})$, the Voronoi relevant vectors of $\mathcal{L}$, to be the minimal set of lattice vectors satisfying $\mathcal{V}(\mathcal{L}) = \cap_{\mathbf{v} \in \text{VR}(\mathcal{L})} H_{\mathbf{v}}$, which we abbreviate to* VR *when the context is clear.*

Since the Voronoi cell is a full dimensional centrally symmetric polytope, the set VR corresponds exactly to the set of lattice vectors inducing facets of $\mathcal{V}$ (i.e. such that $\mathcal{V} \cap \partial H_{\mathbf{v}}$ is $n-1$ dimensional).

Definition 13 (Voronoi Cell Facet)**.** *For each* $\mathbf{v} \in \mathrm{VR}$*, let*

$$F_{\mathbf{v}} = \mathcal{V} \cap \left\{ \mathbf{x} \in \mathbb{R}^n : \langle \mathbf{x}, \mathbf{v} \rangle = \frac{1}{2} \langle \mathbf{v}, \mathbf{v} \rangle \right\},$$

denote the facet of $\mathcal{V}$ *induced by* $\mathbf{v}$*.*

Here we have that

$$\partial \mathcal{V} = \bigcup_{\mathbf{v} \in \mathrm{VR}} F_{\mathbf{v}} \quad \text{and} \quad \mathrm{vol}_{n-1}(\partial \mathcal{V}) = \sum_{\mathbf{v} \in \mathrm{VR}} \mathrm{vol}_{n-1}(F_{\mathbf{v}})$$

since the intersection of distinct facets has affine dimension at most $n-2$. Similarly,

$$\mathcal{V} = \bigcup_{\mathbf{v} \in \mathrm{VR}} \mathrm{conv}(\mathbf{0}, F_{\mathbf{v}}) \quad \text{and} \quad \mathrm{vol}_n(\mathcal{V}) = \sum_{\mathbf{v} \in \mathrm{VR}} \mathrm{vol}_n(\mathrm{conv}(\mathbf{0}, F_{\mathbf{v}})).$$

A central object in this paper will be $\mathcal{L} + \partial \mathcal{V}$, the boundary of the lattice tiling. We shall call $\mathbf{y} + F_{\mathbf{v}}$, for $\mathbf{y} \in \mathcal{L}, \mathbf{v} \in \mathrm{VR}$, a facet fo $\mathcal{L} + \partial \mathcal{V}$. Here, we see that

$$\mathcal{L} + \partial \mathcal{V} = \bigcup_{\mathbf{y} \in \mathcal{L}, \mathbf{v} \in \mathrm{VR}} \mathbf{y} + F_{\mathbf{v}}.$$

Note that each facet is counted twice in the above union, i.e. $\mathbf{y} + F_{\mathbf{v}} = (\mathbf{y} + \mathbf{v}) + F_{-\mathbf{v}}$.

An important theorem of Voronoi classifies the set of Voronoi relevant vectors:

Theorem 14 (Voronoi)**.** *For an n dimensional lattice* $\mathcal{L}$*,* $\mathbf{y} \in \mathcal{L} \setminus \{\mathbf{0}\}$ *is in* $\mathrm{VR}(\mathcal{L})$ *if and only if*

$$\{\pm \mathbf{y}\} = \operatorname*{arg\,min}_{\mathbf{x} \in 2\mathcal{L} + \mathbf{y}} \|\mathbf{x}\|_2.$$

In particular, $|\mathrm{VR}| \leq 2(2^n - 1)$*.*

Here the bound on $|\mathrm{VR}|$ follows from the fact that the map $\mathbf{y} \mapsto \mathbf{y} + 2\mathcal{L}$ from VR to $\mathcal{L}/(2\mathcal{L}) \setminus \{2\mathcal{L}\}$ is 2-to-1. Furthermore, note that each Voronoi relevant vector can be recovered from solutions to CVPs over $2\mathcal{L}$. More precisely, given a basis B for $\mathcal{L}$, each vector in $\mathbf{v} \in \mathrm{VR}$ can be expressed as $B\mathbf{p} - \mathbf{x}$, for some $\mathbf{p} \in \{0,1\}^n \setminus \{\mathbf{0}\}$, and $\mathbf{x} \in \mathsf{CVP}(2\mathcal{L}, B\mathbf{p})$ (we get a Voronoi relevant iff $\mathbf{x}$ is unique up to reflection about $B\mathbf{p}$).

We now list several important and standard properties we will need about the Voronoi cell and relevant vectors. We give a proof for completeness.

Lemma 15. *For an n dimensional lattice* $\mathcal{L}$*:*

1. $\frac{\lambda_1(\mathcal{L})}{2} B_2^n \subseteq \mathcal{V} \subseteq \mu(\mathcal{L}) B_2^n$.
2. $\lambda_1(\mathcal{L}) = \min_{\mathbf{v} \in \mathrm{VR}} \|\mathbf{v}\|_2$.
3. $2\mu(\mathcal{L})/\sqrt{n} \leq \max_{\mathbf{v} \in \mathrm{VR}} \|\mathbf{v}\|_2 \leq 2\mu(\mathcal{L})$

Proof. We prove each of the above in order:

1. Since each vector $\mathbf{y} \in \mathcal{L} \setminus \{\mathbf{0}\}$ satisfies $\|\mathbf{y}\|_2 \geq \lambda_1(\mathcal{L})$, we clearly have that $\lambda_1(\mathcal{L})/2 B_2^n \subseteq H_{\mathbf{y}}$. The inner containment holds for $\mathcal{V}$ since $\mathcal{V} = \cap_{\mathbf{y} \in \mathcal{L} \setminus \{\mathbf{0}\}} H_{\mathbf{y}}$. For the outer containment, note that for any $\mathbf{t} \in \mathcal{V}$, that $\mathbf{0}$ is a closest lattice vector to $\mathbf{t}$. Hence, by definition, $\|\mathbf{t}\|_2 = \|\mathbf{t} - \mathbf{0}\|_2 \leq \mu(\mathcal{L})$ as needed.

2. Since the set VR $\subseteq \mathcal{L} \setminus \{\mathbf{0}\}$, the vectors in VR clearly have length greater than or equal to $\lambda_1(\mathcal{L})$. Next, let $\mathbf{y} \in \mathcal{L} \setminus \{\mathbf{0}\}$ denote a shortest non-zero vector of $\mathcal{L}$. We wish to show that $\mathbf{y} \in$ VR. To do this, by Theorem 14, we need only show that the only vectors of length $\lambda_1(\mathcal{L})$ in $\mathbf{y} + 2\mathcal{L}$ are $\pm\mathbf{y}$. Assume not, then there exists $\mathbf{z} \in \mathbf{y} + 2\mathcal{L}$, such that $\mathbf{z}$ is not collinear with $\mathbf{y}$ having $\|\mathbf{z}\|_2 = \lambda_1(\mathcal{L})$. But then note that $(\mathbf{y}+\mathbf{z})/2 \in \mathcal{L} \setminus \{\mathbf{0}\}$ and $\|(\mathbf{y}+\mathbf{z})/2\|_2 < \lambda_1(\mathcal{L})$, a contradition.

3. For $\mathbf{v} \in$ VR, we remember that $\mathbf{v} = \arg\min_{\mathbf{z} \in 2\mathcal{L}+\mathbf{v}} \|\mathbf{z}\|_2$. In particular, this implies that $\|\mathbf{v}\|_2 \leq \mu(2\mathcal{L}) = 2\mu(\mathcal{L})$ as needed. Since the VR vectors span $\mathbb{R}^n$, we can find linearly independent $\mathbf{v}_1, \ldots, \mathbf{v}_n \in$ VR. By Lemma 10, we have that

$$\mu(\mathcal{L}) \leq (1/2)\sqrt{\sum_{i=1}^{n} \|\mathbf{v}\|_i^2} \leq \sqrt{n}/2 \max_{\mathbf{v} \in \text{VR}} \|\mathbf{v}\|_2 ,$$

as needed.

□

4.2 Convex geometry

A set $K \subseteq \mathbb{R}^n$ is a convex body, if it is convex (i.e. $\mathbf{x}, \mathbf{y} \in K \Rightarrow [\mathbf{x}, \mathbf{y}] \subseteq K$), compact and has non-empty interior. K is symmetric if $K = -K$. For a symmetric convex body $K \subseteq \mathbb{R}^n$, we define the norm (or gauge function) with respect to K by $\|\mathbf{x}\|_K = \inf\{s \geq 0 : \mathbf{x} \in sK\}$, for any $\mathbf{x} \in \mathbb{R}^n$. A function $f : K \to \mathbb{R}$ is convex (concave) if for all $\mathbf{x}, \mathbf{y} \in K$, $\alpha \in [0, 1]$,

$$\alpha f(\mathbf{x}) + (1-\alpha) f(\mathbf{y}) \quad \geq (\leq) \quad f(\alpha\mathbf{x} + (1-\alpha)\mathbf{y}) .$$

For a set $A \subseteq \mathbb{R}^n$, we define its convex hull $\text{conv}(A)$ to be the (inclusion wise) smallest convex set containing A. For two sets $A, B \subseteq \mathbb{R}^n$, we use the notation $\text{conv}(A, B) \stackrel{\text{def}}{=} \text{conv}(A \cup B)$.

For two non-empty measurable sets $A, B \subseteq \mathbb{R}^n$ such that $A + B$ is measurable, the Brunn-Minkowski inequality gives the following fundamental lower bound

$$\text{vol}_n(A+B)^{1/n} \geq \text{vol}_n(A)^{1/n} + \text{vol}_n(B)^{1/n}. \tag{1}$$

Laplace Distributions. We define the Gamma function, $\Gamma(k) = \int_0^\infty x^{k-1}e^{-x}dx$ for $k > 0$. For $k \in \mathbb{N}$, we note that $\Gamma(k) = (k-1)!$. We define the two parameter distribution $\Gamma(k, \theta)$ on $\mathbb{R}$, $k, \theta \geq 0$, to have probability density function $\frac{1}{\theta^k \Gamma(k)} x^{k-1} e^{-x/\theta}$, for $x \in \mathbb{R}$. For $r \sim \Gamma(k, \theta)$, $k \in N$, the moments of r are

$$\mathbb{E}[r^l] = \theta^l \frac{(k+l-1)!}{(k-1)!} \text{ , for } l \in \mathbb{N}.$$

In particular, $\mathbb{E}[r] = k\theta$ and $\mathbb{VAR}[r] = k\theta^2$.

Definition 16 (Laplace Distribution)**.** *We define the probability distribution* Laplace(K, θ)*, with probability density function*

$$f_K^\theta(\mathbf{x}) = \frac{\theta^n}{\text{vol}_n(K) n!} e^{-\|x\|_K/\theta}, \quad \textit{for } \mathbf{x} \in \mathbb{R}^n.$$

Equivalently, a well known and useful fact (which we state without proof) is:

Lemma 17. *$X \sim$ Laplace(K, θ) is identically distributed to rU, where $r \sim \Gamma(n+1, \theta)$ and $U \sim$ Uniform(K) are sampled independently.*

For our purposes, Laplace(K,θ) will serve as a "smoothed" out version of Uniform(K). In particular, letting f denote the probability density function of Laplace(K,θ), for $\mathbf{x},\mathbf{y}\in\mathbb{R}^n$, by the triangle inequality

$$\frac{f_K^\theta(\mathbf{x})}{f_K^\theta(\mathbf{y})}=\frac{e^{-\|\mathbf{x}\|_K/\theta}}{e^{-\|\mathbf{y}\|_K/\theta}}\in[e^{-\|\mathbf{y}-\mathbf{x}\|_K/\theta},e^{\|\mathbf{y}-\mathbf{x}\|_K/\theta}]. \tag{2}$$

Hence, the density varies smoothly as a function of $\|\cdot\|_K$ norm, avoiding the "sharp" boundaries of the uniform measure on K.

Algorithms. A membership oracle O_K for a convex body $K\subseteq\mathbb{R}^n$ is a function satisfying $O_K(\mathbf{x})=1$ if $\mathbf{x}\in K$, and $O_K(\mathbf{x})=0$ otherwise. Most algorithms over convex bodies can be implemented using only a membership oracle with some additional guarantees.

In our CVPP algorithm, we will need to sample nearly uniformly from the Voronoi cell. For this purpose, we will utilize the classic geometric random walk method of Dyer, Frieze, and Kannan [9], which allows for polynomial time near uniform sampling over any convex body.

Theorem 18 (Convex Body Sampler [9])**.** *Let $K\subseteq\mathbb{R}^n$ be a convex body, given my a membership oracle O_K, satisfying $rB_2^n\subseteq K\subseteq RB_2^n$. Then for $\varepsilon>0$, a realisation of a random variable $X\in K$, having total variation distance at most ε from* Uniform(K)*, can be computed using* poly$(n,\log(R/r),\log(1/\varepsilon))$ *arithmetic operations and calls to the membership oracle.*

5 Bounding the Number of Crossings

In this section, we prove bounds on the number of crossings the randomized straight line algorithm induces on the tiling boundary $\mathcal{L}+\partial\mathcal{V}$. For a target $\mathbf{t}$, starting point $\mathbf{x}\in\mathcal{L}$, and perturbation $Z\sim$ Uniform(V), we need to bound the expected number of crossings in phases B and C, that is

$$(B)\ \mathbb{E}[|(\mathcal{L}+\partial\mathcal{V})\cap[\mathbf{x}+Z,\mathbf{t}+Z]|]\qquad(C)\ \mathbb{E}[|(\mathcal{L}+\partial\mathcal{V})\cap[\mathbf{t}+Z,\mathbf{t})|]\,.$$

The phase B bound is given in Section 5.1, and the phase C is given in Section 5.2.

5.1 Phase B estimates

The high level idea of the phase B bound is as follows. To count the number of crossings, we break the segment $[\mathbf{x}+Z,\mathbf{y}+Z]$ into k equal chunks (we will let $k\to\infty$), and simply count the number of chunks which cross at least 1 boundary. By our choice of perturbation, we can show that each point on the segment $[\mathbf{x}+Z,\mathbf{y}+Z]$ is uniformly distributed modulo the lattice, and hence the crossing probability will be indentical on each chunk. In particular, we will get that each crossing probability is exactly the probability that Z "escapes" from $\mathcal{V}$ after moving by $(\mathbf{y}-\mathbf{x})/k$. This measures how close Z tends to be to the boundary of $\mathcal{V}$, and hence corresponds to a certain "directional" surface area to volume ratio.

In the next lemma, we show that the escape probability is reasonably small for any symmetric convex body, when the size of the shift is measured using the norm induced by the body. We shall use this to prove the full phase B crossing bound in Theorem 3.

Lemma 19. *Let $K\subseteq\mathbb{R}^n$ be a centrally symmetric convex body. Then for $Z\sim$ Uniform(K) and $\mathbf{y}\in\mathbb{R}^n$, we have that*

$$\lim_{\varepsilon\to0}\Pr[Z+\varepsilon\mathbf{y}\notin K]/\varepsilon\le(n/2)\|\mathbf{y}\|_K$$

Proof. By applying a linear transformation to K and $\mathbf{y}$, we may assume that $\mathbf{y} = \mathbf{e}_n$. Let $\pi_{n-1} : \mathbb{R}^n \to \mathbb{R}^{n-1}$ denote the projection onto the first $n-1$ coordinates. Define $l : \pi_{n-1}(K) \to \mathbb{R}_+$ as $l(\mathbf{x}) = \text{vol}_1(\{(\mathbf{x}, x_n) : x_n \in \mathbb{R}, (\mathbf{x}, x_n) \in K\})$, i.e. the length of the chord of K passing through $(\mathbf{x}, 0)$ in direction $\mathbf{e}_n$.

For $\mathbf{x} \in \pi_{n-1}(K)$, let $\{(\mathbf{x}, x_n) : x_n \in \mathbb{R}, (\mathbf{x}, x_n) \in K\} = [(\mathbf{x}, a), (\mathbf{x}, b)]$, $a \leq b$, denote its associated chord, where we note that $|b - a| = l(\mathbf{x})$. From here, conditioned on Z landing on this chord, note that $Z + \varepsilon \mathbf{e}_n \notin K$ if and only if Z lies in the half open line segment $((\mathbf{x}, b - \varepsilon), (\mathbf{x}, b)]$. Given this, we have that

$$\begin{aligned}
\lim_{\varepsilon \to 0} \Pr[Z + \varepsilon \mathbf{e}_n \notin K]/\varepsilon &= \lim_{\varepsilon \to 0} (1/\varepsilon) \int_{\pi_{n-1}(K)} \min\{\varepsilon, l(\mathbf{x})\} \frac{\mathrm{d}\mathbf{x}}{\text{vol}_n(K)} \\
&= \lim_{\varepsilon \to 0} \int_{\pi_{n-1}(K)} \min\{1, l(\mathbf{x})/\varepsilon\} \frac{\mathrm{d}\mathbf{x}}{\text{vol}_n(K)} \\
&= \int_{\pi_{n-1}(K)} \frac{\mathrm{d}\mathbf{x}}{\text{vol}_n(K)} = \frac{\text{vol}_{n-1}(\pi_{n-1}(K))}{\text{vol}_n(K)} \,.
\end{aligned} \tag{3}$$

Let $s = 1/\|\mathbf{e}_n\|_K$. Since K is centrally symmetric, $\mathbb{R}\mathbf{e}_n \cap K = [-s\mathbf{e}_n, s\mathbf{e}_n]$ and hence $l(\mathbf{0}) = 2s$. Note that by central symmetry of K, for all $\mathbf{x} \in \pi_{n-1}(K)$, $l(\mathbf{x}) = l(-\mathbf{x})$. Since K is convex, the function l is concave on $\pi_{n-1}(K)$, and hence

$$\max_{\mathbf{x} \in \pi_{n-1}(K)} l(\mathbf{x}) = \max_{\mathbf{x} \in \pi_{n-1}(K)} \frac{1}{2} l(\mathbf{x}) + \frac{1}{2} l(-\mathbf{x}) \leq \max_{\mathbf{x} \in \pi_{n-1}(K)} l(\mathbf{0}) = 2s.$$

Let $K' = \{(\mathbf{x}, x_n) : \mathbf{x} \in \pi_{n-1}(K), 0 \leq x_n \leq l(\mathbf{x})\}$. By concavity of l, it is easy to see that K' is also a convex set. Furthermore, note that K' has exactly that same chord lengths as K in direction $\mathbf{e}_n$, and hence $\text{vol}_n(K') = \text{vol}_n(K)$. For $a \in \mathbb{R}$, let $K'_a = K' \cap \{(\mathbf{x}, a) : \mathbf{x} \in \mathbb{R}^{n-1}\}$. Here $\mathbb{R}\mathbf{e}_n \cap K' = [\mathbf{0}, 2s\mathbf{e}_n]$, and hence $K_a \neq \emptyset$, $\forall a \in [0, 2s]$. Therefore $K_a = \emptyset$ for $a > 2s$, since the maximum chord length is $l(\mathbf{0}) = 2s$, as well as for $a < 0$. By construction of K', we see that $K'_0 = \pi_{n-1}(K) \times \{0\}$, and hence $\text{vol}_{n-1}(K'_0) = \text{vol}_{n-1}(\pi_{n-1}(K))$.

Given (3), to prove the Lemma, it now suffices to show that

$$\frac{\text{vol}_{n-1}(\pi_{n-1}(K))}{\text{vol}_n(K)} = \frac{\text{vol}_{n-1}(\pi_{n-1}(K))}{\text{vol}_n(K')} \leq (n/2)\|\mathbf{e}_n\|_K \,.$$

For $a \in [0, 2s]$, by convexity of K' and the Brunn-Minkowski inequality on $\mathbb{R}^{n-1}$, we have that

$$\begin{aligned}
\text{vol}_{n-1}(K'_a)^{\frac{1}{n-1}} &\geq \text{vol}_{n-1}((1 - \frac{a}{2s})K'_0 + \frac{a}{2s}K'_{2s})^{\frac{1}{n-1}} \\
&\geq (1 - \frac{a}{2s})\text{vol}_{n-1}(K'_0)^{\frac{1}{n-1}} + \frac{a}{2s}\text{vol}_{n-1}(K'_{2s})^{\frac{1}{n-1}} \\
&\geq (1 - \frac{a}{2s})\text{vol}_{n-1}(\pi_{n-1}(K))^{\frac{1}{n-1}} \,.
\end{aligned}$$

Therefore, we have that

$$\begin{aligned}
\text{vol}_n(K') &= \int_0^{2s} \text{vol}_{n-1}(K'_a)\mathrm{da} \geq \text{vol}_{n-1}(\pi_{n-1}(K)) \int_0^{2s} \left(1 - \frac{a}{2s}\right)^{n-1} \mathrm{da} \\
&= \text{vol}_{n-1}(\pi_{n-1}(K))(2s) \int_0^1 (1-a)^{n-1} \mathrm{da} = \text{vol}_{n-1}(\pi_{n-1}(K))(2s)/n \\
&= \frac{2\text{vol}_{n-1}(\pi_{n-1}(K))}{n\|\mathbf{e}\|_n},
\end{aligned}$$

as needed. □

5.1.1 Proof of Theorem 3 (Phase B crossing bound)

Proof. Note first that the sets $(\mathcal{L}+\partial\mathcal{V})\cap[\mathbf{x}+Z,\mathbf{y}+Z]$ and $(\mathcal{L}+\partial\mathcal{V})\cap[\mathbf{x}+Z,\mathbf{y}+Z)$ agree unless $\mathbf{y}+Z\in\mathcal{L}+\partial\mathcal{V}$. Given that this event happens with probability 0 (as $\mathcal{L}+\partial\mathcal{V}$ has n dimensional Lebesgue measure 0), we get that

$$\mathbb{E}_Z[|(\mathcal{L}+\partial\mathcal{V})\cap[\mathbf{x}+Z,\mathbf{y}+Z]|]=\mathbb{E}_Z[|(\mathcal{L}+\partial\mathcal{V})\cap[\mathbf{x}+Z,\mathbf{y}+Z)|]\,.$$

We now bound the expectation on the right hand side. For $s\in[0,1]$, define the random variable $\ell(s)=(1-s)\mathbf{x}+s\mathbf{y}+Z$. Let A_j^k, $0\le j<2^k$, denote the event that

$$|(\mathcal{L}+\partial\mathcal{V})\cap[\ell(j/2^k),\ell((j+1)/2^k))|\ge 1\quad\Leftrightarrow\quad|(\mathcal{L}+\partial\mathcal{V})\cap[\ell(j/2^k),\ell(j/2^k)+(\mathbf{y}-\mathbf{x})/2^k)|\ge 1\,.$$

Clearly, we have that

$$|(\mathcal{L}+\partial\mathcal{V})\cap[\ell(0),\ell(1))|=\lim_{k\to\infty}\sum_{j=0}^{2^k-1}A_j^k\,.$$

By the monotone convergence theorem, we get that

$$\mathbb{E}_Z[|(\mathcal{L}+\partial\mathcal{V})\cap[\ell(0),\ell(1))|]=\lim_{k\to\infty}\sum_{j=0}^{2^k-1}\Pr[A_j^k=1]\,. \tag{4}$$

Since $\mathcal{L}+\partial\mathcal{V}$ is by definition invariant under lattice shifts, we see that $\Pr[A_j^k=1]$ depends only on the distribution of $\ell(j/2^k)\pmod{\mathcal{L}}$. Given that $Z\pmod{\mathcal{L}}\sim\text{uniform}(\mathbb{R}^n/\mathcal{L})$ and that $\mathbb{R}^n/\mathcal{L}$ is shift invariant, we have that $\ell(j/2^k)\pmod{\mathcal{L}}\sim\text{uniform}(\mathbb{R}^n/\mathcal{L})$. In particular, this implies that $\Pr[A_0^k]=\cdots=\Pr[A_{2^k}^k]$, and hence by Lemma 19

$$\begin{aligned}\lim_{k\to\infty}\sum_{j=0}^{2^k-1}\Pr[A_j^k=1]&=\lim_{k\to\infty}2^k\Pr[A_0^k=1]\\&=\lim_{k\to\infty}2^k\Pr[Z+(\mathbf{y}-\mathbf{x})/2^k\notin\mathcal{V}]\le(n/2)\|\mathbf{y}-\mathbf{x}\|_{\mathcal{V}}\,,\end{aligned} \tag{5}$$

as needed. The result follows by combining (4) and (5). □

5.2 Phase C estimates

As mentioned previously in the paper, our techniques will not be sufficient to fully bound the number of phase C crossings. However, we will use be able to give bounds for a truncation of the phase C path, that is for $\alpha\in(0,1]$, we will bound

$$\mathbb{E}[|(\mathcal{L}+\partial\mathcal{V})\cap[\mathbf{t}+Z,\mathbf{t}+\alpha Z]|]\,.$$

We will give a bound of $O(n\ln 1/\alpha)$ for the above crossings in Theorem 4.

For the proof strategy, we follow the approach as phase B in terms of bounding the crossing probability on infinitessimal chunks of $[\mathbf{t}+Z,\mathbf{t}+\alpha Z]$. However, the implementation of this strategy will be completely different here, since the points along the segment no longer have the same distribution modulo the lattice. In particular, as $\alpha\to 0$, the distributions get more concentrated, and hence we lose the effects of the randomness. This loss will be surprisingly mild however, as evidenced by the claimed $\ln(1/\alpha)$ dependence.

For the infinitessimal probabilities, it will be convenient to parametrize the segment $[\mathbf{t}+Z,\mathbf{t}+\alpha Z]$ differently than in phase B. In particular, we use $\mathbf{t}+Z/s$, for $s\in[1,1/\alpha]$. From here, note that

$$\Pr[(\mathcal{L}+\partial\mathcal{V})\cap[\mathbf{t}+Z/s,\mathbf{t}+Z/(s+\varepsilon)]\neq\emptyset]=\Pr[Z\in\cup_{\gamma\in[s,s+\varepsilon]}\gamma(\mathcal{L}-\mathbf{t}+\partial\mathcal{V})]\,. \tag{6}$$

Taking the limit as $\varepsilon \to 0$, we express the infinitessimal probability as a certain weighted surface area integral over $s(\mathcal{L} - \mathbf{t} + \partial\mathcal{V})$ (see Lemma 27).

In the same spirit as phase B, we will attempt to relate the surface integral to a nicely bounded integral over all of space. To help us in this task, we will rely on a technical trick to "smooth out" the distribution of Z. More precisely, we will replace the perturbation $Z \sim \text{Uniform}(\mathcal{V})$ by the perturbation $X \sim \text{Laplace}(\mathcal{V}, \theta)$, for an appropriate choice of θ. For the relationship between both types of perturbatoins, we will use the representation of X as rZ, where $r \sim \Gamma(n+1, \theta)$. We will choose θ so that r is concentrated in the interval $[1, 1 + \frac{1}{n}]$, which will insure that the number of crossings for X and Z are roughly the same.

The benefit of the Laplace perturbation for us will be as follows. Since it varies much more smoothly than the uniform distribution (which has "sharp boundaries"), it will allow us to make the analysis of the surface integral entirely local by using the periodic structure of $s(\mathcal{L} - \mathbf{t} + \partial\mathcal{V})$. In particular, we will be able to relate the surface integral over each tile $s(\mathbf{y} - \mathbf{t} + \partial\mathcal{V})$, $\mathbf{y} \in \mathcal{L}$, to a specific integral over each cone $s(\mathbf{y} + \text{conv}(\mathbf{0}, F_{\mathbf{v}}))$, $\forall \mathbf{v} \in \text{VR}$, making up the tile. Under the uniform distribution, the probability density over each tile can be challenging to analyze, since the tile may only be partially contained in $\mathcal{V}$. However, under the Laplace distribution, we know that over $s(\mathbf{y} + \mathcal{V})$ the density can vary by at most $e^{\pm s/\theta}$ (see Equation 2 in the Preliminaries).

The integral over $\mathbb{R}^n$ we end up using to control the surface integral over $s(\mathcal{L} - \mathbf{t} + \partial\mathcal{V})$ turns out the be rather natural. At all scales, we simply use the integral representation of $\mathbb{E}[\|X\|_{\mathcal{V}}] = n\theta$ (see Lemma 25). In particular, as $s \to \infty$, for the appropriate choice of θ, this will allow us to bound the surface integral over $s(\mathcal{L} - \mathbf{t} + \partial\mathcal{V})$ by $O(n/s)$. Integrating this bound from 1 to $1/\alpha$ yields the claimed $O(n \ln(1/\alpha))$ bound on the number of crossings.

This section is organized as follows. In subsection 5.2.1, we relate the number of crossings for uniform and Laplace perturbations. In subsection 5.2.2, we bound the number of crossings for Laplace perturbations. Lastly, in subsection 5.2.3, we combine the estimates from the previous subsections to give the full phase C in Theorem 4.

5.2.1 Converting Uniform Perturbations to Laplace Perturbations

In this section, we show that the number of crossings induced by uniform perturbations can be controlled by the number of crossings induced by Laplace perturbation.

We define $\theta_n = \frac{1}{(n+1) - \sqrt{2(n+1)}}$, $\gamma_n = (1 + \frac{2\sqrt{2}}{\sqrt{n+1} - \sqrt{2}})^{-1}$ for use in the rest of this section.

The following Lemma shows the $\Gamma(n+1, \theta)$ distribution is concentrated in a small interval above 1 for the appropriate choice of θ. This will be used in Lemma 21 to relate the number of crossings between the uniform and Laplace perturbations.

Lemma 20. *For $r \sim \Gamma(n+1, \theta_n)$, $n \geq 2$, we have that*

$$\Pr[r \in [1, 1 + \frac{2\sqrt{2}}{\sqrt{n+1} - \sqrt{2}}]] \geq \frac{1}{2}$$

Proof. Remember that $\mathbb{E}[r] = (n+1)\theta_n$ and that $\mathbb{VAR}[r] = (n+1)\theta_n^2$. Letting $\sigma = \sqrt{\mathbb{VAR}[r]}$, by Chebyshev's inequality

$$\Pr[|r - \mathbb{E}[r]| \geq \sqrt{2}\sigma] \leq \frac{\mathbb{VAR}[r]}{2\sigma^2} = \frac{1}{2}$$

The result now follows from the identities

$$\mathbb{E}[r] - \sqrt{2}\sigma = ((n+1) - \sqrt{2(n+1)})\theta_n = 1$$

$$\mathbb{E}[r] + \sqrt{2}\sigma = ((n+1) + \sqrt{2(n+1)})\theta_n = 1 + \frac{2\sqrt{2}}{\sqrt{n+1} - \sqrt{2}}$$

□

Lemma 21. *Let $\mathcal{L}$ be an n-dimensional lattice, $n \geq 2$, and $\mathbf{t} \in \mathbb{R}^n$. Then for $\alpha \in [0,1]$, $Z \sim \text{Uniform}(\mathcal{V})$ and $X \sim \text{Laplace}(\mathcal{V}, \theta_n)$, we have that*

$$\mathbb{E}_Z[|(\mathcal{L}+\partial\mathcal{V}) \cap [Z+\mathbf{t}, \alpha Z+\mathbf{t}]|] \leq 2\,\mathbb{E}_X[|(\mathcal{L}+\partial\mathcal{V}) \cap [X+\mathbf{t}, \gamma_n \alpha X+\mathbf{t}]]$$

where $\gamma_n = (1+\frac{2\sqrt{2}}{\sqrt{n+1}-\sqrt{2}})^{-1}$.

Proof. We shall use the fact that X is identically distributed to rZ where $r \sim \Gamma(n+1, \theta_n)$ is sampled independently from Z. Conditioned on any value of Z, the following inclusion holds

$$[Z+\mathbf{t}, \alpha Z+\mathbf{t}) \subseteq [rZ+\mathbf{t}, \gamma_n \alpha r Z+\mathbf{t})$$

as long as $r \in [1, \gamma_n^{-1}] = [1, 1+\frac{2\sqrt{2}}{\sqrt{n+1}-\sqrt{2}}]$. By Lemma 20, we get that

$$\begin{aligned}
\mathbb{E}_X[|(\mathcal{L}+\partial\mathcal{V}) \cap [X+\mathbf{t}, \gamma_n\alpha X+\mathbf{t}]|] &= \mathbb{E}_Z[\ \mathbb{E}_r[|(\mathcal{L}+\partial\mathcal{V}) \cap [rZ+\mathbf{t}, \gamma_n\alpha rZ+\mathbf{t}]|]\ ] \\
&\geq \mathbb{E}_Z[|(\mathcal{L}+\partial\mathcal{V}) \cap [Z+\mathbf{t}, \alpha Z+\mathbf{t}]|\,\Pr[r \in [1, \gamma_n^{-1}]]] \\
&\geq \frac{1}{2}\,\mathbb{E}_Z[|(\mathcal{L}+\partial\mathcal{V}) \cap [Z+\mathbf{t}, \alpha Z+\mathbf{t}]|],
\end{aligned}$$

as needed. □

5.2.2 Bounding the Number of Crossing for Laplace Perturbations

In this section, we bound the number of crossings induced by Laplace perturbations. The expression for the infinitessimal crossing probabilities is given in Lemma 23, the bound on the surface area integral over $s(\mathcal{L}-\mathbf{t}+\partial\mathcal{V})$ to $\mathbb{E}[\|X\|_{\mathcal{V}}]$ is given in Lemma 25, and the full phase C Laplace crossing bound is given in Theorem 27.

For $\mathbf{t} \in \mathbb{R}^n$, and $s > 0$, the set $s(\mathcal{L}-\mathbf{t}+\partial\mathcal{V})$ is a shifted and scaled version of the tiling boundary $\mathcal{L}+\partial\mathcal{V}$. For $\mathbf{y} \in \mathcal{L}$, and $\mathbf{v} \in \text{VR}$, we will call $s(\mathbf{y}-\mathbf{t}+F_{\mathbf{v}})$ a facet of $s(\mathcal{L}-\mathbf{t}+\partial\mathcal{V})$.

Definition 22 (Tiling boundary normals)**.** *We define the function $\eta : (\mathcal{L}-\mathbf{t}+\partial\mathcal{V}) \to S^{n-1}$ as follows. For $\mathbf{x} \in (\mathcal{L}-\mathbf{t}+\partial\mathcal{V})$, choose the lexicographically minimal $\mathbf{v} \in \text{VR}$ such that $\exists \mathbf{y} \in \mathcal{L}$ satisfying $\mathbf{x} \in (\mathbf{y}-\mathbf{t}+F_{\mathbf{v}})$. Finally, define $\eta(\mathbf{x}) = \mathbf{v}/\|\mathbf{v}\|_2$.*

Note that for $\mathbf{x} \in s(\mathcal{L}-\mathbf{t}+\partial\mathcal{V})$, $\eta(\mathbf{x}/s)$ is a unit normal to a facet of $s(\mathcal{L}-\mathbf{t}+\partial\mathcal{V})$ containing $\mathbf{x}$. Furthermore, the subset of points in $s(\mathcal{L}-\mathbf{t}+F_{\mathbf{v}})$ having more than one containing facet has $n-1$ dimensional measure 0, and hence can be ignored from the perspective of integration over $s(\mathcal{L}-\mathbf{t}+\partial\mathcal{V})$.

The following lemma gives the expression for the infinitessimal crossing probabilities.

Lemma 23. *For $\alpha \in (0,1]$, and $X \sim \text{Laplace}(\mathcal{V}, \theta)$, we have that*

$$\mathbb{E}[|(\mathcal{L}+\partial\mathcal{V}) \cap [X+\mathbf{t}, \alpha X+\mathbf{t}]|] = \int_1^{1/\alpha} \int_{s(\mathcal{L}-\mathbf{t}+\partial\mathcal{V})} |\,\langle \eta(\mathbf{x}/s), \mathbf{x}/s\rangle\,| f_{\mathcal{V}}^{\theta}(\mathbf{x}) \text{dvol}_{n-1}(\mathbf{x}) \text{ds}\,.$$

Proof. Firstly, shifting by $-\mathbf{t}$ on both sides, we get that

$$\mathbb{E}[|(\mathcal{L}+\partial\mathcal{V}) \cap [X+\mathbf{t}, \alpha X+\mathbf{t}]|] = \mathbb{E}[|(\mathcal{L}-\mathbf{t}+\partial\mathcal{V}) \cap [X, \alpha X]|].$$

From here, we first decompose the expected number of intersections by summing over all facets. This yields

$$\mathbb{E}[|(\mathcal{L}-\mathbf{t}+\partial\mathcal{V}) \cap [X, \alpha X]|] = \frac{1}{2} \sum_{\mathbf{y}\in\mathcal{L}, \mathbf{v}\in\text{VR}} \mathbb{E}[|(\mathbf{y}-\mathbf{t}+F_{\mathbf{v}}) \cap [X, \alpha X]|]. \tag{7}$$

The factor $1/2$ above accounts for the fact that we count each facet twice, i.e. $\mathbf{y}-\mathbf{t}+F_{\mathbf{v}}$ and $(\mathbf{y}+\mathbf{v})-\mathbf{t}+F_{-\mathbf{v}}$. Secondly, note that the intersections we count more than twice in the above decomposition correspond to a countable number of lines passing through at most $n-2$ dimensional faces, and hence have n dimensional Lebesgue measure 0. The equality in Equation (7) thus follows.

If we restrict to one facet $\mathbf{y}-\mathbf{t}+F_{\mathbf{v}}$, for some $\mathbf{y}\in\mathcal{L}$, $\mathbf{v}\in \mathrm{VR}$, we note that the line segment $[X,\alpha X]$ crosses the facet $\mathbf{y}-\mathbf{t}+F_{\mathbf{v}}$ at most once with probability 1. Hence, we get that

$$\begin{aligned}\mathbb{E}[|(\mathbf{y}-\mathbf{t}+F_{\mathbf{v}})\cap[X,\alpha X]|] &= \Pr[(\mathbf{y}-\mathbf{t}+F_{\mathbf{v}})\cap[X,\alpha X]\neq\emptyset]\\ &= \Pr[X\in\cup_{s\in[1,\frac{1}{\alpha}]}s(\mathbf{y}-\mathbf{t}+F_{\mathbf{v}})]\\ &= \int_{\cup_{s\in[1,\frac{1}{\alpha}]}s(\mathbf{y}-\mathbf{t}+F_{\mathbf{v}})} f_{\mathcal{V}}^{\theta}(\mathbf{x})\mathrm{d}\mathbf{x}\,.\end{aligned} \tag{8}$$

Let $r=\langle \mathbf{v}/\|\mathbf{v}\|_2, \mathbf{y}-\mathbf{t}+F_{\mathbf{v}}\rangle$, noting the inner product with $\mathbf{v}$ (and hence $\hat{\mathbf{v}}$) is constant over $F_{\mathbf{v}}$. By possibly switching $\mathbf{v}$ to $-\mathbf{v}$ and $\mathbf{y}$ to $\mathbf{y}+\mathbf{v}$ (which maintains the facet), we may assume that $r\geq 0$. Notice that by construction, for any $\mathbf{x}$ in the (relative) interior of $s(\mathbf{y}-\mathbf{t}+F_{\mathbf{v}})$, we get that $r=|\langle \eta(\mathbf{x}/s),\mathbf{x}/s\rangle|$, since then there is a unique facet of $s(\mathcal{L}-\mathbf{t}+\partial\mathcal{V})$ containing $\mathbf{x}$. Integrating first in the in the direction $\mathbf{v}$, we get that

$$\begin{aligned}&\int_{\cup_{s\in[1,\frac{1}{\alpha}]}s(\mathbf{y}-\mathbf{t}+F_{\mathbf{v}})} f_{\mathcal{V}}^{\theta}(\mathbf{x})\mathrm{d}\mathbf{x} = \int_{r}^{r/\alpha}\int_{(s/r)(\mathbf{y}-\mathbf{t}+F_{\mathbf{v}})} f_{\mathcal{V}}^{\theta}(\mathbf{x})\mathrm{dvol}_{n-1}(\mathbf{x})\mathrm{d}s =\\ &\int_{1}^{1/\alpha}\int_{s(\mathbf{y}-\mathbf{t}+F_{\mathbf{v}})} r f_{\mathcal{V}}^{\theta}(\mathbf{x})\mathrm{dvol}_{n-1}(\mathbf{x})\mathrm{d}s = \int_{1}^{1/\alpha}\int_{s(\mathbf{y}-\mathbf{t}+F_{\mathbf{v}})} |\langle \eta(\mathbf{x}/s),\mathbf{x}/s\rangle| f_{\mathcal{V}}^{\theta}(\mathbf{x})\mathrm{dvol}_{n-1}(\mathbf{x})\mathrm{d}s\,.\end{aligned} \tag{9}$$

Note that we use the $n-1$ dimensional Lebesgue measure to integrate over $s(\mathbf{y}-\mathbf{t}+F_{\mathbf{v}})$ since it is embedded in $\mathbb{R}^n$. If $r=0$, note that the set $\cup_{s\in[1,\frac{1}{\alpha}]}s(\mathbf{y}-\mathbf{t}+F_{\mathbf{v}})$ is $n-1$ dimensional and hence has measure (and probability) 0. This is still satisfied by the last expression in (9), and hence the identity is still valid in this case.

Putting everything together, combining Equation (7),(9), we get that

$$\begin{aligned}\mathbb{E}[|(\mathcal{L}-\mathbf{t}+\partial\mathcal{V})\cap[X,\alpha X]|] &= \frac{1}{2}\sum_{\mathbf{y}\in\mathcal{L},\mathbf{v}\in\mathrm{VR}}\int_{1}^{1/\alpha}\int_{s(\mathbf{y}-\mathbf{t}+F_{\mathbf{v}})} |\langle \eta(\mathbf{x}/s),\mathbf{x}/s\rangle| f_{\mathcal{V}}^{\theta}(\mathbf{x})\mathrm{dvol}_{n-1}(\mathbf{x})\mathrm{d}s =\\ &= \int_{1}^{1/\alpha}\int_{s(\mathcal{L}-\mathbf{t}+\partial\mathcal{V})} |\langle \eta(\mathbf{x}/s),\mathbf{x}/s\rangle| f_{\mathcal{V}}^{\theta}(\mathbf{x})\mathrm{dvol}_{n-1}(\mathbf{x})\mathrm{d}s\,,\end{aligned}$$

as needed. □

The lower bound given in the following Lemma will be needed in the proof of Lemma 25.

Lemma 24. *For $a,b,c,d\in\mathbb{R}$, $c\leq d$, we have that*

$$\int_{c}^{d}|a+bh|\mathrm{d}h \geq (\sqrt{2}-1)(d-c)\max\{|(a+bc)|,|a+bd|\}\,.$$

Proof. Firstly, we note that

$$\int_{c}^{d}|a+bh|\mathrm{d}h = (d-c)\int_{0}^{1}|a+b(c+(d-c)h)|\mathrm{d}h = (d-c)\int_{0}^{1}|(a+bc)+b(d-c)h|\mathrm{d}h\,,$$

hence it suffices to prove the inequality when $c=0,d=1$. After this reduction, by possibly applying the change of variables $h\leftarrow 1-h$, we may assume that $|a|\geq|a+b|$. Next, by changing the signs of a,b, we may assume that $a\geq 0$. Hence, it remains to prove the inequality

$$\int_{0}^{1}|a+bh|\mathrm{d}h \geq a(\sqrt{2}-1) \tag{10}$$

under the assumption that $a \geq |a+b|$, or equivalently $a \geq 0$ and $-2a \leq b \leq 0$. Notice that if $a = 0$ or $b = 0$, the above inequality is trivially true. If $a, b \neq 0$, then dividing inequality 10 by a, we reduce to the case where $a = 1$, $-2 \leq b < 0$. Letting $\alpha = -1/b$, we have that $\alpha \in [1/2, \infty)$. From here, we get that

$$\int_0^1 |1 + hb| \mathrm{dh} = \int_0^1 |1 - h/\alpha| \mathrm{dh} = (1/2)(\alpha + (1-\alpha)^2/\alpha)$$

The derivative of the above expression is $1 - \frac{1}{2\alpha^2}$. The expression is thus minimized for $\alpha = \frac{1}{\sqrt{2}} > 1/2$, and the result follows by plugging in this value. □

We now prove the bound on the surface integral in terms of the expectation $\mathbb{E}[\|X\|_{\mathcal{V}}]$.

Lemma 25. *For $s \geq 1$ and $X \sim \mathrm{Laplace}(\mathcal{V}, \theta)$, we have that*

$$\int_{s(\mathcal{L}-\mathbf{t}+\partial\mathcal{V})} |\langle \eta(\mathbf{x}/s), \mathbf{x}/s \rangle| f_{\mathcal{V}}^{\theta}(\mathbf{x}) \mathrm{dvol}_{n-1}(\mathbf{x}) \leq c \max\left\{\frac{n}{s^2}, \frac{1}{\theta s}\right\} \mathbb{E}[\|X\|_{\mathcal{V}}] = c \max\left\{\frac{n^2\theta}{s^2}, \frac{n}{s}\right\},$$

for $c = \frac{e^2}{2(\sqrt{2}-1)} \leq 9$.

Proof. We first prove the equality on the right hand side. We remember that X is identically distributed to rZ where $r \sim \Gamma(n+1, \theta)$ and $Z \sim \mathrm{Uniform}(\mathcal{V})$. From here, we have that

$$\begin{aligned}
\mathbb{E}[\|X\|_{\mathcal{V}}] &= \mathbb{E}[\|rZ\|_{\mathcal{V}}] = \mathbb{E}[r]\,\mathbb{E}[\|Z\|_{\mathcal{V}}] \\
&= (n+1)\theta \int_0^1 \Pr[\|Z\|_{\mathcal{V}} \geq s] \mathrm{ds} \\
&= (n+1)\theta \int_0^1 (1 - s^n) \mathrm{ds} = (n+1)\theta \left(\frac{n}{n+1}\right) = n\theta\,,
\end{aligned}$$

as needed.

We now prove the first inequality. To prove the bound, we write the integral expressing $\mathbb{E}[\|X\|_{\mathcal{V}}]$ over the cells of $s(\mathcal{L} - \mathbf{t} + \partial\mathcal{V})$, and compare the integral over each cell to the corresponding boundary integral. To begin

$$\begin{aligned}
\mathbb{E}[\|X\|_{\mathcal{V}}] &= \int_{\mathbb{R}^n} \|\mathbf{x}\|_{\mathcal{V}} f_{\mathcal{V}}^{\theta}(\mathbf{x}) \mathrm{d}\mathbf{x} = \sum_{\mathbf{y} \in \mathcal{L}} \int_{s(\mathbf{y}-\mathbf{t})+s\mathcal{V}} \|\mathbf{x}\|_{\mathcal{V}} f_{\mathcal{V}}^{\theta}(\mathbf{x}) \mathrm{d}\mathbf{x} \\
&= \sum_{\mathbf{y} \in \mathcal{L}, \mathbf{v} \in \mathrm{VR}} \int_{s(\mathbf{y}-\mathbf{t})+\mathrm{conv}(\mathbf{0}, sF_{\mathbf{v}})} \|\mathbf{x}\|_{\mathcal{V}} f_{\mathcal{V}}^{\theta}(\mathbf{x}) \mathrm{d}\mathbf{x}\,.
\end{aligned} \tag{11}$$

Fix $\mathbf{y} \in \mathcal{L}$ and $\mathbf{v} \in \mathcal{V}$ in the above sum. Noting that $\langle \mathbf{v}/\|\mathbf{v}\|_2, sF_{\mathbf{v}} \rangle = s\|\mathbf{v}\|_2/2$ by construction, and integrating first in the direction $\mathbf{v}$, we get that

$$\begin{aligned}
&\int_{s(\mathbf{y}-\mathbf{t})+\mathrm{conv}(\mathbf{0}, sF_{\mathbf{v}})} \|\mathbf{x}\|_{\mathcal{V}} f_{\mathcal{V}}^{\theta}(\mathbf{x}) \mathrm{d}\mathbf{x} = \\
&\int_0^{s\|\mathbf{v}\|_2/2} \int_{\frac{2h}{s\|\mathbf{v}\|_2}(sF_{\mathbf{v}})} \|s(\mathbf{y}-\mathbf{t}) + \mathbf{x}\|_{\mathcal{V}} f_{\mathcal{V}}^{\theta}(s(\mathbf{y}-\mathbf{t}) + \mathbf{x}) \mathrm{dvol}_{n-1}(\mathbf{x}) \mathrm{dh}\,.
\end{aligned} \tag{12}$$

In the above, we use the $n-1$ dimensional Lebesgue measure to integrate over $\frac{2h}{s\|\mathbf{v}\|_2} F_{\mathbf{v}}$ since it is embedded in $\mathbb{R}^n$ (we also do this for ease of notation). Setting $\beta = \frac{2h}{s\|\mathbf{v}\|_2}$, note that $\beta \in [0,1]$. In equation (12), β represents the convex combination between $\mathbf{0}$ and $sF_{\mathbf{v}}$, that is $\mathrm{conv}(\mathbf{0}, F_{\mathbf{v}}) = \bigcup_{\beta \in [0,1]} \beta s F_{\mathbf{v}}$.

Performing a change of variables, Equation (12) simplifies to

$$\begin{aligned}
&\int_0^1 \int_{h(sF_{\mathbf{v}})} (s\|\mathbf{v}\|_2/2)\|s(\mathbf{y}-\mathbf{t})+\mathbf{x}\|_{\mathcal{V}} f_{\mathcal{V}}^{\theta}(s(\mathbf{y}-\mathbf{t})+\mathbf{x})\mathrm{dvol}_{n-1}(\mathbf{x})\mathrm{dh} = \\
&\int_{sF_{\mathbf{v}}} \int_0^1 (s\|\mathbf{v}\|_2/2)\|s(\mathbf{y}-\mathbf{t})+h\mathbf{x}\|_{\mathcal{V}} f_{\mathcal{V}}^{\theta}(s(\mathbf{y}-\mathbf{t})+h\mathbf{x})h^{n-1}\mathrm{dh}\,\mathrm{dvol}_{n-1}(\mathbf{x}) = \\
&\int_{sF_{\mathbf{v}}} \int_0^1 (s\|\mathbf{v}\|_2/2)\|s(\mathbf{y}-\mathbf{t})+(1-h)\mathbf{x}\|_{\mathcal{V}} f_{\mathcal{V}}^{\theta}(s(\mathbf{y}-\mathbf{t})+(1-h)\mathbf{x})(1-h)^{n-1}\mathrm{dh}\,\mathrm{dvol}_{n-1}(\mathbf{x})\,.
\end{aligned} \tag{13}$$

From here we note that

$$\begin{aligned}
(s\|\mathbf{v}\|_2/2)\|s(\mathbf{y}-\mathbf{t})+(1-h)\mathbf{x}\|_{\mathcal{V}} &= (s\|\mathbf{v}\|_2/2)\max_{\mathbf{w}\in \mathrm{VR}} \left|\frac{2\langle \mathbf{w}, s(\mathbf{y}-\mathbf{t})+(1-h)\mathbf{x}\rangle}{\langle \mathbf{w},\mathbf{w}\rangle}\right| \\
&\geq (s\|\mathbf{v}\|_2/2)\left|\frac{2\langle \mathbf{v}, s(\mathbf{y}-\mathbf{t})+(1-h)\mathbf{x}\rangle}{\langle \mathbf{v},\mathbf{v}\rangle}\right| \\
&= s^2\left|\langle \mathbf{v}/\|\mathbf{v}\|_2, (\mathbf{y}-\mathbf{t})+(1-h)\mathbf{x}/s\rangle\right|.
\end{aligned} \tag{14}$$

From inequality (14), we have that the expression in equation (13) is greater than or equal to

$$\int_{sF_{\mathbf{v}}} s^2 \int_0^1 \left|\langle \mathbf{v}/\|\mathbf{v}\|_2, (\mathbf{y}-\mathbf{t})+(1-h)\mathbf{x}/s\rangle\right| f_{\mathcal{V}}^{\theta}(s(\mathbf{y}-\mathbf{t})+(1-h)\mathbf{x})(1-h)^{n-1}\mathrm{dh}\,\mathrm{dvol}_{n-1}(\mathbf{x})\,. \tag{15}$$

To compare to the surface integral, we now lower bound the inner integral.

Claim 26. *For $\|\mathbf{x}\|_{\mathcal{V}} \leq s$, we have that*

$$\begin{aligned}
&\int_0^1 \left|\langle \mathbf{v}/\|\mathbf{v}\|_2, (\mathbf{y}-\mathbf{t})+(1-h)\mathbf{x}/s\rangle\right| f_{\mathcal{V}}^{\theta}(s(\mathbf{y}-\mathbf{t})+(1-h)\mathbf{x})(1-h)^{n-1}\mathrm{dh} \\
&\geq\ e^{-2}(\sqrt{2}-1)\min\left\{\frac{1}{n},\frac{\theta}{s}\right\}\ \left|\langle \mathbf{v}/\|\mathbf{v}\|_2, (\mathbf{y}-\mathbf{t})+\mathbf{x}/s\rangle\right| f_{\mathcal{V}}^{\theta}(s(\mathbf{y}-\mathbf{t})+\mathbf{x})
\end{aligned}$$

Proof. Note that for $0 \leq h \leq \min\left\{\frac{1}{n},\frac{\theta}{s}\right\}$, we have that

$$\begin{aligned}
f_{\mathcal{V}}^{\theta}(s(\mathbf{y}-\mathbf{t})+(1-h)\mathbf{x})(1-h)^{n-1} &\geq f_{\mathcal{V}}^{\theta}(s(\mathbf{y}-\mathbf{t})+\mathbf{x})e^{-\|h\mathbf{x}\|_{\mathcal{V}}/\theta}(1-h)^{n-1} \\
&\geq f_{\mathcal{V}}^{\theta}(s(\mathbf{y}-\mathbf{t})+\mathbf{x})e^{-1}(1-1/n)^{n-1} \geq e^{-2} f_{\mathcal{V}}^{\theta}(s(\mathbf{y}-\mathbf{t})+\mathbf{x})\,.
\end{aligned}$$

Hence, using the above and Lemma 24, we have that

$$\begin{aligned}
&\int_0^1 \left|\langle \mathbf{v}/\|\mathbf{v}\|_2, (\mathbf{y}-\mathbf{t})+(1-h)\mathbf{x}/s\rangle\right| f_{\mathcal{V}}^{\theta}(s(\mathbf{y}-\mathbf{t})+(1-h)\mathbf{x})(1-h)^{n-1}\mathrm{dh} \\
&\geq e^{-2} f_{\mathcal{V}}^{\theta}(s(\mathbf{y}-\mathbf{t})+\mathbf{x})\int_0^{\min\left\{\frac{1}{n},\frac{\theta}{s}\right\}} \left|\langle \mathbf{v}/\|\mathbf{v}\|_2, (\mathbf{y}-\mathbf{t})+(1-h)\mathbf{x}/s\rangle\right| \mathrm{dh} \\
&\geq e^{-2}(\sqrt{2}-1)\min\left\{\frac{1}{n},\frac{\theta}{s}\right\}\ \left|\langle \mathbf{v}/\|\mathbf{v}\|_2, (\mathbf{y}-\mathbf{t})+\mathbf{x}/s\rangle\right| f_{\mathcal{V}}^{\theta}(s(\mathbf{y}-\mathbf{t})+\mathbf{x})\,,
\end{aligned}$$

as needed. □

Given Claim 26, we get that expression (15) is greater than or equal to

$$\begin{aligned}
&\int_{sF_{\mathbf{v}}} s^2 e^{-2}(\sqrt{2}-1)\min\left\{\frac{1}{n},\frac{\theta}{s}\right\}\ \left|\langle \mathbf{v}/\|\mathbf{v}\|_2, (\mathbf{y}-\mathbf{t})+\mathbf{x}/s\rangle\right| f_{\mathcal{V}}^{\theta}(s(\mathbf{y}-\mathbf{t})+\mathbf{x})\mathrm{dvol}_{n-1}(\mathbf{x}) = \\
&e^{-2}(\sqrt{2}-1)\min\left\{\frac{s^2}{n}, s\theta\right\}\int_{s(\mathbf{y}-\mathbf{t}+F_{\mathbf{v}})} \left|\langle \eta_s(\mathbf{x}/s), \mathbf{x}/s\rangle\right| f_{\mathcal{V}}^{\theta}(\mathbf{x})\mathrm{dvol}_{n-1}(\mathbf{x})\,.
\end{aligned}$$

Putting everything together, combining the above with equation (11), we get that

$$\begin{aligned}\mathbb{E}[\|X\|_{\mathcal{V}}] &= \sum_{\mathbf{y}\in\mathcal{L},\mathbf{v}\in\mathrm{VR}} \int_{s(\mathbf{y}-\mathbf{t})+\mathrm{conv}(\mathbf{0},sF_{\mathbf{v}})} \|\mathbf{x}\|_{\mathcal{V}} f_{\mathcal{V}}^{\theta}(\mathbf{x})\mathrm{d}\mathbf{x} \\ &\geq e^{-2}(\sqrt{2}-1)\min\left\{\frac{s^2}{n}, s\theta\right\} \sum_{\mathbf{y}\in\mathcal{L},\mathbf{v}\in\mathrm{VR}} \int_{s(\mathbf{y}-\mathbf{t}+F_{\mathbf{v}})} |\langle \eta_s(\mathbf{x}/s), \mathbf{x}/s\rangle| f_{\mathcal{V}}^{\theta}(\mathbf{x})\mathrm{dvol}_{n-1}(\mathbf{x}) = \\ &= 2e^{-2}(\sqrt{2}-1)\min\left\{\frac{s^2}{n}, s\theta\right\} \int_{s(\mathcal{L}-\mathbf{t}+\partial\mathcal{V})} |\langle \eta_s(\mathbf{x}/s), \mathbf{x}/s\rangle| f_{\mathcal{V}}^{\theta}(\mathbf{x})\mathrm{dvol}_{n-1}(\mathbf{x}) ,\end{aligned}$$

where the last equality follows since each facet in $s(\mathcal{L}-\mathbf{t}+\partial\mathcal{V})$ is counted twice. The lemma thus follows. □

The following gives the full phase C bound for Laplace perturbations.

Theorem 27. *For $\alpha \in (0,1]$ and $X \sim \mathrm{Laplace}(\mathcal{V},\theta)$, we have that*

$$\mathbb{E}[|(\mathcal{L}+\partial\mathcal{V}) \cap [X+\mathbf{t}, \alpha X+\mathbf{t}]|] \leq cn\left(\frac{n\theta}{2}\left(1-\frac{1}{s^*}\right)+\ln\left(\frac{1}{\alpha s^*}\right)\right)$$

where $c = \frac{e^2}{2(\sqrt{2}-1)} \leq 9$ and $s^* = \max\{1, n\theta\}$.

Proof. Using Lemmas 23 and 25, we have that

$$\begin{aligned}\mathbb{E}[|(\mathcal{L}+\partial\mathcal{V}) \cap [X+\mathbf{t}, \alpha X+\mathbf{t}]|] &= \int_1^{1/\alpha} \int_{s(\mathcal{L}-\mathbf{t}+\partial\mathcal{V})} |\langle \eta(\mathbf{x}/s), \mathbf{x}/s\rangle| f_{\mathcal{V}}^{\theta}(\mathbf{x})\mathrm{dvol}_{n-1}(\mathbf{x})\mathrm{ds} \\ &\leq c\int_1^{1/\alpha} \max\left\{\frac{n^2\theta}{s^2}, \frac{n}{s}\right\}\mathrm{ds} = c\int_1^{s^*} \frac{n^2\theta}{s^2}\mathrm{ds} + c\int_{s^*}^{1/\alpha} \frac{n}{s}\mathrm{ds} \\ &= cn^2\theta\left(1/2-\frac{1}{s^*}\right)+cn\ln\left(\frac{1}{\alpha s^*}\right) \\ &= cn\left(\frac{n\theta}{2}\left(1-\frac{1}{s^*}\right)+\ln\left(\frac{1}{\alpha s^*}\right)\right) ,\end{aligned}$$

as needed. □

5.2.3 Proof of Theorem 4 (Phase C crossing bound)

Proof. We recall that $\theta_n = \frac{1}{(n+1)-\sqrt{2(n+1)}}$ and $\gamma_n = \left(1+\frac{2\sqrt{2}}{\sqrt{n+1}-\sqrt{2}}\right)^{-1}$. Note that $n\theta_n > 1$.

By Lemma 21 and Theorem 27, for $X \sim \mathrm{Laplace}(\mathcal{V},\theta_n)$, we have that

$$\begin{aligned}\mathbb{E}[|(\mathcal{L}+\partial\mathcal{V}) \cap [Z+\mathbf{t}, \alpha Z+\mathbf{t}]|] &\leq 2\,\mathbb{E}[|(\mathcal{L}+\partial\mathcal{V}) \cap [X+\mathbf{t}, \gamma_n\alpha X+\mathbf{t}]|] \\ &\leq 2cn\left(\frac{n\theta_n}{2}\left(1-\frac{1}{n\theta_n}\right)+\ln\left(\frac{1}{\alpha\gamma_n n\theta_n}\right)\right) \\ &\leq \frac{e^2}{\sqrt{2}-1} n(2+\ln(4/\alpha)) \text{ , for } n \geq 2 ,\end{aligned}$$

as needed. □

6 Missings proofs from Section 3

Proof of Lemma 2 (Randomized Straight Line Complexity). We recall the three phases of the randomized straight line algorithm:

(A) Move from $\mathbf{x}$ to $\mathbf{x} + Z$.

(B) Follow the sequence of Voronoi cells from $\mathbf{x} + Z$ to $\mathbf{y} + Z$.

(C) Follow the sequence of Voronoi cells from $\mathbf{y} + Z$ to $\mathbf{y}$.

Characterizing the Path Length: We will show that with probability 1, the length of the path on $\mathcal{G}$ induced by the three phases is

$$|(\mathcal{L} + \partial\mathcal{V}) \cap [\mathbf{x} + Z, \mathbf{t} + Z]| + |(\mathcal{L} + \partial\mathcal{V}) \cap [\mathbf{t} + Z, \mathbf{t})| \,.$$

Firstly, note that since $Z \in \mathcal{V}$ and $\mathbf{x} \in \mathcal{L}$, $\mathbf{x} + Z$ and $\mathbf{x}$ lie in the same Voronoi cell $\mathbf{x} + Z$, and hence phase A corresponds to the trivial path $\mathbf{x}$. Hence, we need only worry about the number of edges induced by phases B and C.

The following claim will establish the structure of a generic intersection pattern with the tiling boundary, which will be necessary for establishing the basic properties of the path.

Claim 28. *With probability* 1, *the path* $[\mathbf{x} + Z, \mathbf{t} + Z] \cup [\mathbf{t} + Z, \mathbf{t})$ *only intersects* $\mathcal{L} + \partial\mathcal{V}$ *in the relative interior of its facets. Furthermore, with probability* 1, *the intersection consists of isolated points, and* $\mathbf{x} + Z, \mathbf{t} + Z \notin \mathcal{L} + \partial\mathcal{V}$.

Proof. We prove the first part. Let $C_1, \ldots, C_k$ denote the $n-2$ dimensional faces of $\mathcal{V}$. Note that the probability of not hitting $\mathcal{L} + \partial\mathcal{V}$ in the relative interior of it facets, can be expressed as

$$\begin{aligned} &\Pr[([\mathbf{x} + Z, \mathbf{t} + Z] \cup [\mathbf{t} + Z, \mathbf{t})) \cap \left(\cup_{i \in [k]} \mathcal{L} + C_i\right) \neq \emptyset] \leq \\ &\sum_{\mathbf{y} \in \mathcal{L}, i \in [k]} \Pr[[\mathbf{x} + Z, \mathbf{t} + Z] \cap (\mathbf{y} + C_i) \neq \emptyset] + \Pr[[\mathbf{t} + Z, \mathbf{t}) \cap (\mathbf{y} + C_i) \neq \emptyset] \,. \end{aligned} \tag{16}$$

Here the last inequality is valid since $\mathcal{L}$ is countable. Analyzing each term separately, we see that

$$\Pr[[\mathbf{x} + Z, \mathbf{t} + Z] \cap (\mathbf{y} + C_i) \neq \emptyset] = \Pr[Z \in \mathbf{y} + C_i - [\mathbf{x}, \mathbf{t}]] = 0.$$

To justify the last equality, note that since C_i is $n-2$ dimensional, $\mathbf{y} + C_i - [\mathbf{x}, \mathbf{t}]$ is at most $n-1$ dimensional (since the line segment can only add 1 dimension). Therefore $\mathbf{y} + C_i - [\mathbf{x}, \mathbf{t}]$ has n dimensional Lebesgue measure 0, and in particular probability 0 with respect to Uniform$(\mathcal{V})$. Next, we have that

$$\Pr[[\mathbf{t} + Z, \mathbf{t}) \cap (\mathbf{y} + C_i) \neq \emptyset] = \Pr[Z \in \cup_{s>1} s(\mathbf{y} + C_i - \mathbf{t})] = 0 \,,$$

where the last equality follows since $\cup_{s>1} s(\mathbf{y} + C_i - \mathbf{t})$ is at most $n-1$ dimensional. Hence the probability in (16) is 0, as needed.

We now prove the second part. Note that if the path $[\mathbf{x} + Z, \mathbf{t} + Z] \cup [\mathbf{t} + Z, \mathbf{t})$ does not intersect $\mathcal{L} + \partial\mathcal{V}$ in isolated points (i.e. the intersection contains a non-trivial interval), then either $[\mathbf{x} + Z, \mathbf{t} + Z]$ or $[\mathbf{t} + Z, \mathbf{t})$ must intersect some facet of $\mathcal{L} + \partial\mathcal{V}$ in a least 2 distinct points.

Let $F_{\mathbf{v}}$ be the facet of $\mathcal{V}$ induced by $\mathbf{v} \in \text{VR}$. If $[\mathbf{t} + Z, \mathbf{t})$ intersects $\mathbf{y} + F_{\mathbf{v}}$, for some $\mathbf{y} \in \mathcal{L}$, in two distinct points then we must have that $\langle \mathbf{v}, Z \rangle = 0$. Since $\Pr[\cup_{\mathbf{v} \in \text{VR}} \{\langle \mathbf{v}, Z \rangle = 0\}] = 0$, this event happens with probability 0. Next, note that $[\mathbf{x} + Z, \mathbf{t} + Z]$ intersects $\mathbf{y} + F_{\mathbf{v}}$ in two distinct points, if and only if $\langle \mathbf{v}, \mathbf{x} - \mathbf{t} \rangle = 0$ and $\langle \mathbf{v}, \mathbf{Z} \rangle = \langle \mathbf{v}, \mathbf{y} + \mathbf{v}/2 \rangle$. But then, the probability of this happening for any facet can be bounded by

$$\Pr[\cup_{\mathbf{y} \in \mathcal{L}, \mathbf{v} \in \text{VR}} \{\langle \mathbf{v}, Z \rangle = \langle \mathbf{v}, \mathbf{y} + \mathbf{v}/2 \rangle\}] = 0$$

since $\mathcal{L} \times \mathrm{VR}$ is countable.

For the last part, note that since $\mathcal{L} + \partial\mathcal{V}$ is the union of $n-1$ dimensional pieces, $\Pr[\mathbf{x} + Z \in \mathcal{L} + \partial\mathcal{V}] + \Pr[\mathbf{t} + Z \in \mathcal{L} + \partial\mathcal{V}] = 0$.

The claim thus follows. □

Conditioning on the intersection structure given in claim 28, we now describe the associated path on $\mathcal{G}$. Let $\mathbf{p}_1, \ldots, \mathbf{p}_k$ denote the points in $([\mathbf{x} + Z, \mathbf{t} + Z] \cup [\mathbf{t} + Z, \mathbf{t})) \cap (\mathcal{L} + \partial\mathcal{V})$ ordered in order of appearance on the path $[\mathbf{x} + Z, \mathbf{t} + Z] \cup [\mathbf{t} + Z, \mathbf{t})$ from left to right. Letting $\mathbf{p}_{k+1} = \mathbf{t}$, let $\mathbf{y}_i \in \mathcal{L}$, $1 \le i \le k$, denote the center of the unique Voronoi cell in $\mathcal{L} + \mathcal{V}$ containing the interval $[\mathbf{p}_i, \mathbf{p}_{i+1}]$. Note that the existence of $\mathbf{y}_i$ is guaranteed since the Voronoi cells in the tiling $\mathcal{L} + \mathcal{V}$ are interior disjoint, and the open segment $(\mathbf{p}_i, \mathbf{p}_{i+1})$ lies in the interior of some Voronoi cell by convexity of the cells.

Letting $\mathbf{y}_0 = \mathbf{x}$, we now claim that $\mathbf{y}_0, \mathbf{y}_1, \ldots, \mathbf{y}_k$ form a valid path in $\mathcal{G}$. To begin, we first establish that $\mathbf{y}_i \neq \mathbf{y}_{i+1}$, $0 \le i \le k$. Firstly, since $\mathbf{x} + Z \notin \mathcal{L} + \partial\mathcal{V}$, we have that Z is in the interior of $\mathcal{V}$, and hence the ray starting at Z in the direction of $\mathbf{p}_1$ exits $\mathbf{x} + \mathcal{V}$ at $\mathbf{p}_1$ and never returns (by convexity of $\mathcal{V}$). Furthermore, since $\mathbf{p}_1 \neq \mathbf{t} + Z$, the Voronoi cell $\mathbf{y}_1 + \mathcal{V}$ must contain a non-trivial interval on this ray starting at $\mathbf{p}_1$, i.e. $[\mathbf{p}_1, \mathbf{p}_2]$, and hence $\mathbf{y}_1 \neq \mathbf{x}$. Indeed, for the remaining cases, the argument follows in the same way as long as the Voronoi cell $\mathbf{y}_{i+1} + \mathcal{V}$ contains a non-trivial interval of the ray exiting $\mathbf{y}_i + \mathcal{V}$. Note that this is guaranteed by the assumption that $\mathbf{t} + Z \notin \partial\mathcal{V}$ and by the fact that none of the $\mathbf{p}_i$s equals $\mathbf{t}$. Hence $\mathbf{y}_i \neq \mathbf{y}_{i+1}$, $0 \le i \le k$, as needed.

Next, note that each p_i, $i \in [k]$, belongs to the relative interior of some facet of $\mathcal{L} + \partial\mathcal{V}$. Furthermore, by construction $p_i \in \mathbf{y}_{i-1} + \partial\mathcal{V}$ and $p_i \in \mathbf{y}_i + \partial\mathcal{V}$. Since the relative interior of facets of $\mathcal{L} + \partial\mathcal{V}$ touch exactly two adjacent Voronoi cells, and since $\mathbf{y}_{i-1} \neq \mathbf{y}_i$, we must have that $\mathbf{p}_i \in \mathbf{y}_{i-1} + F_{\mathbf{v}}$, where $\mathbf{v} = \mathbf{y}_i - \mathbf{y}_{i-1} \in \mathrm{VR}$. Hence the path $\mathbf{y}_0, \mathbf{y}_1, \ldots, \mathbf{y}_k$ is valid in $\mathcal{G}$ as claimed.

From here, note that the length of is indeed $k = |([\mathbf{x} + Z, \mathbf{t} + Z] \cup [\mathbf{t} + Z, \mathbf{t}) \cap (\mathcal{L} + \partial\mathcal{V})|$. Since this holds with probability 1, we get that the expected path length is

$$\mathbb{E}[|(\mathcal{L} + \partial\mathcal{V}) \cap [\mathbf{x} + Z, \mathbf{t} + Z]|] + \mathbb{E}[|(\mathcal{L} + \partial\mathcal{V}) \cap [\mathbf{t} + Z, \mathbf{t})|] \,.$$

as needed.

Computing the Path: We now explain how to compute each edge of the path using $O(n|\mathrm{VR}|)$ arithmetic operations, conditioning on the conclusions of Claim 28.

In constructing the path, we will in fact compute the intersection points $\mathbf{p}_1, \ldots, \mathbf{p}_k$ as above, and the lattice points $\mathbf{y}_1, \ldots, \mathbf{y}_k$. As one would expect, this computation is broken up in phase B and C, corresponding to computing the intersection / lattice points for $[\mathbf{x} + Z, \mathbf{t} + Z]$ in phase B, followed by the intersection / lattice points from $[\mathbf{t} + Z, \mathbf{t})$ in Phase C.

For each phase, we will use a generic line following procedure that given vectors $\mathbf{a}, \mathbf{b} \in \mathbb{R}^n$, and a starting lattice point $\mathbf{z} \in \mathcal{L}$, such that $\mathbf{a} \in \mathbf{z} + \mathcal{V}$, follows the path of Voronoi cells along the line segment $[\mathbf{a}, \mathbf{b})$, and outputs a lattice vector $\mathbf{w} \in \mathcal{L}$ satisfying $\mathbf{b} \in \mathbf{w} + \mathcal{V}$. To implement phase B, we initialize the procedure with $\mathbf{x} + Z, \mathbf{t} + Z$ and starting point $\mathbf{x}$. For phase C, we give it $\mathbf{t} + Z, \mathbf{t}$ and the output of phase B as the starting point.

We describe the line following procedure. Let $\ell(\alpha) = (1 - \alpha)\mathbf{a} + \alpha\mathbf{b}$, for $\alpha \in [0, 1]$, i.e. the parametrization of $[\mathbf{a} + Z, \mathbf{b} + Z]$ as a function of time. The procedure will have a variable for α, which will be set at its bounds at the beginning and end of the procedure, starting at 0 ending at ≥ 1, and in intermediate steps will correspond to an intersection point. We will also have a variable $\mathbf{w} \in \mathcal{L}$, corresponding to the current Voronoi cell center. We will maintain the invariant that $\ell(\alpha) \in \mathbf{w} + \mathcal{V}$, and furthermore that $\ell(\alpha) \in \mathbf{w} + \partial\mathcal{V}$ for $\alpha \in (0, 1)$.

The line following algorithm is as follows:

Described in words, each loop iteration does the following: given the current Voronoi cell $\mathbf{w} + \mathcal{V}$, and the entering intersection point $\ell(\alpha)$ of the line segment $[\mathbf{a}, \mathbf{b}]$ with respect to $\mathbf{w} + \mathcal{V}$, we first

Data: $\mathbf{a}, \mathbf{b} \in \mathbb{R}^n$, $\mathbf{z} \in \mathcal{L}$, $\mathbf{a} \in \mathbf{z} + \mathcal{V}$
Result: $\mathbf{w} \in \mathcal{L}$ such that $\mathbf{b} \in \mathbf{w} + \mathcal{V}$
$\mathbf{w} \leftarrow \mathbf{z}$, $\mathbf{e} \leftarrow 0$, $\alpha \leftarrow 0$
$\mathrm{VR}' \leftarrow \{\mathbf{v} \in \mathrm{VR} : \langle \mathbf{v}, \mathbf{b} - \mathbf{a} \rangle > 0\}$
repeat
  $\mathbf{w} \leftarrow \mathbf{w} + \mathbf{e}$
  $\mathbf{e} \leftarrow \arg\min_{\mathbf{v} \in \mathrm{VR}'} \frac{\langle \mathbf{v}, \mathbf{v}/2 + \mathbf{w} - \mathbf{a} \rangle}{\langle \mathbf{b} - \mathbf{a}, \mathbf{v} \rangle}$
  $\alpha \leftarrow \frac{\langle \mathbf{e}, \mathbf{e}/2 + \mathbf{w} - \mathbf{a} \rangle}{\langle \mathbf{b} - \mathbf{a}, \mathbf{e} \rangle}$
until $\alpha \geq 1$
return $\mathbf{w}$

compute the exiting intersection point $\ell(\alpha')$, $\alpha' > \alpha$, and the exiting facet $\mathbf{w} + F_{\mathbf{e}}$. If $\alpha' \geq 1$, we know that $\mathbf{b} \in [\ell(\alpha), \ell(\alpha')] \subseteq \mathbf{w} + \mathcal{V}$, and hence we may return $\mathbf{w}$. Otherwise, we move to the center of the Voronoi cell sharing the facet $\mathbf{w} + F_{\mathbf{e}}$ opposite $\mathbf{w}$.

To verify the correctness, we need only show that the line $[\mathbf{a}, \mathbf{b}]$ indeed exits $\mathbf{w} + \mathcal{V}$ through the facet $\mathbf{w} + F_{\mathbf{e}}$ at the end of each iteration. Note that by our invariant $\ell(\alpha) \in \mathbf{w} + \mathcal{V}$ at the beginning of the iteration, and hence

$$\begin{aligned} \langle \mathbf{v}, \ell(\alpha) - \mathbf{w} \rangle \leq \frac{1}{2} \langle \mathbf{v}, \mathbf{v} \rangle, \quad \forall \mathbf{v} \in \mathrm{VR} \quad \Leftrightarrow \\ \langle \mathbf{v}, (1-\alpha)\mathbf{a} + \alpha \mathbf{b} - \mathbf{w} \rangle \leq \frac{1}{2} \langle \mathbf{v}, \mathbf{v} \rangle, \quad \forall \mathbf{v} \in \mathrm{VR} \quad \Leftrightarrow \\ \alpha \langle \mathbf{v}, \mathbf{b} - \mathbf{a} \rangle \leq \langle \mathbf{v}, \mathbf{v}/2 - \mathbf{a} + \mathbf{w} \rangle, \quad \forall \mathbf{v} \in \mathrm{VR} \end{aligned} \tag{17}$$

Since we move along the line segment $[\mathbf{a}, \mathbf{b}]$ by increasing α, i.e. going from $\mathbf{a}$ to $\mathbf{b}$, note that the only constraints that can be eventually violated as we increase α are those for which $\langle \mathbf{v}, \mathbf{b} - \mathbf{a} \rangle > 0$. Hence, in finding the first violated constraint (i.e. exiting facet), we may restrict our attention to the subset of Voronoi relevant vectors $\mathrm{VR}' = \{\mathbf{v} \in \mathrm{VR} : \langle \mathbf{v}, \mathbf{b} - \mathbf{a} \rangle > 0\}$ as done in the algorithm.

From (17), we see that we do not to cross any facet $\mathbf{w} + F_{\mathbf{v}}$, $\mathbf{v} \in \mathrm{VR}'$, as long as

$$\alpha \leq \frac{\langle \mathbf{v}, \mathbf{v}/2 - \mathbf{a} + \mathbf{w} \rangle}{\langle \mathbf{v}, \mathbf{b} - \mathbf{a} \rangle}, \quad \forall \mathbf{v} \in \mathrm{VR}' .$$

Hence the first facet we violate must be induced by

$$\mathbf{e} = \arg\min_{\mathbf{v} \in \mathrm{VR}'} \frac{\langle \mathbf{v}, \mathbf{v}/2 - \mathbf{a} + \mathbf{w} \rangle}{\langle \mathbf{v}, \mathbf{b} - \mathbf{a} \rangle} . \tag{18}$$

Letting $\alpha' = \frac{\langle \mathbf{e}, \mathbf{e}/2 - \mathbf{a} + \mathbf{w} \rangle}{\langle \mathbf{e}, \mathbf{b} - \mathbf{a} \rangle}$, we see that $\ell(\alpha') \in \mathbf{w} + F_{\mathbf{e}}$ is the correctly computed exiting point (corresponding to $\ell(\alpha)$ at the end of the loop iteration), and that $\mathbf{w} + F_{\mathbf{e}}$ is the exiting facet. Since the facet $\mathbf{w} + F_{\mathbf{e}}$ is shared by $(\mathbf{w} + \mathbf{e}) + \mathcal{V}$, we see that $\ell(\alpha') \in (\mathbf{w} + \mathbf{e}) + \partial\mathcal{V}$, and hence the invariant is maintained in the next iteration. The line following algorithm is thus correct.

Notice that each iteration of the line following procedure clearly requires at most $O(n|\mathrm{VR}|)$ arithmetic operations. We note that the conclusions of Claim 28 are only needed to ensure that each iteration of the path finding procedure can be associated with exactly one intersection point in $([\mathbf{x} + Z, \mathbf{t} + Z] \cup [\mathbf{t} + Z, \mathbf{t})) \cap (\mathcal{L} + \partial\mathcal{V})$. In particular, it assures that the minimizer in (18) is unique. This concludes the proof of the Lemma. □

Proof of Lemma 7 (Bit length bound). Clearly,

$$\bar{q} \leq (\prod_{ij} q_{ij}^B)(\prod_i q_i^{\mathbf{t}}) \Rightarrow \log_2 \bar{q} \leq \sum_{ij} \log_2(q_{ij}^B) + \sum_i q_i^{\mathbf{t}} . \tag{19}$$

Hence $\log_2 \bar{q}$ is smaller than the sum of encoding sizes of the denominators of the entries of B and $\mathbf{t}$. Next, it is well known that $\mu(\mathcal{L}) \leq \frac{1}{2}\sqrt{\sum_{ij} B_{ij}^2}$ (see for example [5]). From here, we get that

$$\begin{aligned} \log_2 \mu(\mathcal{L}) &\leq \log_2 \left(\sqrt{\sum_{ij} B_{ij}^2} \right) \leq \log_2 \left(\sqrt{\sum_{ij} (p_{ij}^B)^2} \right) \leq \log_2 \left(\sqrt{\prod_{ij} ((p_{ij}^B)^2 + 1)} \right) \\ &= \sum_{ij} \log_2 \left(\sqrt{(p_{ij}^B)^2 + 1} \right) \leq \sum_{ij} \log_2 \left(|p_{ij}^B| + 1 \right) \end{aligned} \tag{20}$$

Hence $\log_2 \mu(\mathcal{L})$ is less than the sum of encoding sizes of the numerators of the entries in B. The bound $\log_2(\bar{q}\mu(\mathcal{L})) \leq \text{enc}(B) + \text{enc}(\mathbf{t})$ now follows by adding (19),(20).

We now bound $\log_2(\mu(\mathcal{L})/\lambda_1(\mathcal{L}))$. Letting $\tilde{q} = \prod_{ij} q_{ij}^B$, note that

$$\tilde{q}\lambda_1(\mathcal{L}) = \lambda_1(\tilde{q}\mathcal{L}) \geq \lambda_1(\mathbb{Z}^n) = 1.$$

Therefore $1/\lambda_1(\mathcal{L}) \leq \tilde{q}$. Since $\log_2 \tilde{q} \leq \sum_{ij} \log_2(q_{ij}^B)$ and $\log_2(\mu(\mathcal{L})/\lambda_1(\mathcal{L})) \leq \log_2(\tilde{q}\mu(\mathcal{L}))$, combining with (20) we get that $\log_2(\mu(\mathcal{L})/\lambda_1(\mathcal{L})) \leq \text{enc}(B)$ as needed. □

7 Open Problems

Our work here raises a number of natural questions. Firstly, given the improvement for CVPP, it is natural to wonder whether any of the insights developed here can be used to improve the complexity upper bound for CVP. As mentioned previously, this would seem to require new techniques, and we leave this as an open problem.

Secondly, while we now have a number of methods to navigate over the Voronoi graph, we have no lower bounds on the lengths of the path they create. In particular, it is entirely possible that either the MV path or the simple deterministic straight line path, also yield short paths on the Voronoi graph. Hence, showing either strong lower bounds for these methods or new upper bounds is an interesting open problem. In this vein, as mentioned previously, we do not know whether the expected number of iterations for the randomized straight line procedure is inherently weakly polynomial. We leave this as an open problem.

References

[1] Dorit Aharonov and Oded Regev. Lattice problems in NP ∩ coNP. *J. ACM*, 52(5):749–765, 2005. Preliminary version in FOCS 2004.

[2] Miklós Ajtai, Ravi Kumar, and D. Sivakumar. A sieve algorithm for the shortest lattice vector problem. In *STOC*, pages 601–610, 2001.

[3] Miklós Ajtai, Ravi Kumar, and D. Sivakumar. Sampling short lattice vectors and the closest lattice vector problem. In *IEEE Conference on Computational Complexity*, pages 53–57, 2002.

[4] Mikhail Alekhnovich, Subhash Khot, Guy Kindler, and Nisheeth K. Vishnoi. Hardness of approximating the closest vector problem with pre-processing. *Computational Complexity*, 20(4):741–753, 2011.

[5] László Babai. On Lovász' lattice reduction and the nearest lattice point problem. *Combinatorica*, 6(1):1–13, 1986. Preliminary version in STACS 1985.

[6] Zvika Brakerski and Vinod Vaikuntanathan. Lattice-based fhe as secure as pke. In *Proceedings of the 5th Conference on Innovations in Theoretical Computer Science*, pages 1–12, 2014.

[7] D. Dadush, N. Stephen Davidowitz, and O. Regev. On the closest vector problem with a distance guarantee. In *Proceedings of the Conference on Computational Complexity (CCC)*, 2014.

[8] Rudi de Buda. Some optimal codes have structure. *IEEE Journal on Selected Areas in Communications*, 7(6):893–899, 1989.

[9] M. E. Dyer, A. M. Frieze, and R. Kannan. A random polynomial-time algorithm for approximating the volume of convex bodies. *J. ACM*, 38(1):1–17, 1991. Preliminary version in STOC 1989.

[10] Uri Erez and Ram Zamir. Achieving 1/2 log (1+SNR) on the AWGN channel with lattice encoding and decoding. *IEEE Transactions on Information Theory*, 50(10):2293–2314, 2004.

[11] Uriel Feige and Daniele Micciancio. The inapproximability of lattice and coding problems with preprocessing. *Journal of Computer and System Sciences*, 69(1):45–67, 2004. Preliminary version in CCC 2002.

[12] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *STOC*, pages 169–178, 2009.

[13] Craig Gentry. Toward basing fully homomorphic encryption on worst-case hardness. In *CRYPTO*, pages 116–137, 2010.

[14] Guillaume Hanrot and Damien Stehlé. Improved analysis of Kannan's shortest lattice vector algorithm. In *Proceedings of the 27th annual international cryptology conference on Advances in cryptology*, CRYPTO'07, pages 170–186, Berlin, Heidelberg, 2007. Springer-Verlag.

[15] Bettina Helfrich. Algorithms to construct Minkowski reduced and Hermite reduced lattice bases. *Theor. Comput. Sci.*, 41:125–139, December 1985.

[16] Ravi Kannan. Minkowski's convex body theorem and integer programming. *Mathematics of operations research*, 12(3):415–440, August 1987. 1987.

[17] Subhash Khot, Preyas Popat, and Nisheeth K. Vishnoi. $2^{\log^{1-\varepsilon} n}$ hardness for the closest vector problem with preprocessing. In *STOC*, pages 277–288, 2012.

[18] J. C. Lagarias, Hendrik W. Lenstra Jr., and Claus-Peter Schnorr. Korkin-Zolotarev bases and successive minima of a lattice and its reciprocal lattice. *Combinatorica*, 10(4):333–348, 1990.

[19] J. C. Lagarias and Andrew M. Odlyzko. Solving low-density subset sum problems. *J. ACM*, 32(1):229–246, 1985.

[20] Arjen K. Lenstra, Hendrik W. Lenstra, Jr., and László Lovász. Factoring polynomials with rational coefficients. *Mathematische Annalen*, 261(4):515–534, December 1982.

[21] Hendrik W. Lenstra. Integer programming with a fixed number of variables. *Mathematics of Operations Research*, 8(4):538–548, November 1983.

[22] L. Lovász and S. Vempala. Fast algorithms for logconcave functions: Sampling, rounding, integration and optimization. In *FOCS '06: Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science*, pages 57–68, Washington, DC, USA, 2006. IEEE Computer Society.

[23] R.G. McKilliam, A. Grant, and I.V. Clarkson. Finding a closest point in lattices of Voronoi's first kind. Arxiv report 1405.7014, 2014.

[24] D. Micciancio and P. Voulgaris. A deterministic single exponential time algorithm for most lattice problems based on voronoi cell computations. *SIAM Journal on Computing*, 42(3):1364–1391, 2013. Preliminary version in STOC 2010.

[25] Daniele Micciancio. The hardness of the closest vector problem with preprocessing. *IEEE Transactions on Information Theory*, 47(3):1212–1215, 2001.

[26] Daniele Micciancio and Michael Walter. Fast lattice point enumeration with minimal overhead. In *To appear in the Proceedings of the ACM-SIAM Symposium on Discrete Algorithms*, 2015.

[27] Oded Regev. Improved inapproximability of lattice and coding problems with preprocessing. *IEEE Transactions on Information Theory*, 50(9):2031–2037, 2004. Preliminary version in CCC'03.

[28] Naftali Sommer, Meir Feder, and Ofir Shalvi. Finding the closest lattice point by iterative slicing. *SIAM Journal on Discrete Mathematics*, 23(2):715–731, 2009.

Résumé en français

Approches géométriques et duales en ordonnancement cumulatif

Problématique

Ce travail s'inscrit dans le domaine de l'optimisation mathématique, et plus précisément en ordonnancement à base de programmation par contraintes. Ceci consiste à déterminer les dates de début et de fin d'exécution de tâches (ces dates constituent les variables de décision du problème d'optimisation), tout en satisfaisant à des contraintes temporelles et de ressources, et en optimisant une fonction objectif.

En programmation par contraintes, un problème de cette nature est résolu par une exploration arborescente des domaines des variables de décision par séparation-évaluation. De plus, à chaque noeud est effectuée une phase de propagation, c'est-à-dire que l'on élimine des valeurs des domaines des variables de décision en vérifiant des conditions nécessaires pour que les différentes contraintes soient satisfaites.

Dans ce cadre, la contrainte de ressource la plus fréquemment rencontrée est la cumulative. Elle permet en effet de modéliser des processus se déroulant de manière parallèle, mais en consommant pendant leur exécution une ressource partagée disponible en quantité finie (par exemple des machines ou un budget). Propager cette contrainte de manière efficace est donc d'une importance cruciale pour l'efficacité pratique d'un moteur d'ordonnancement à base de programmation par contraintes.

Nous étudions dans cette thèse la contrainte cumulative en nous servant d'outils rarement utilisés en programmation par contraintes (analyse polyédrale, dualité de la programmation linéaire, dualité de la géométrie projective). À l'aide de ces outils, nous proposons deux contributions pour le domaine : le renforcement cumulatif, et le Raisonnement Énergétique en $O(n^2 \log n)$.

Contributions

Renforcement cumulatif

Nous proposons une reformulation de la contrainte cumulative, c'est-à-dire la génération de contraintes redondantes plus serrées, ce qui permet une propagation plus forte, sans perdre bien entendu de solution faisable.

Cette technique est couramment utilisée en programmation linéaire entière (génération de coupes), mais il s'agit de l'un des tous premiers exemples d'une contrainte globale redondante en programmation par contraintes.

Le calcul de cette reformulation repose sur un programme linéaire dont la taille dépend uniquement de la capacité de la ressource mais pas du nombre de tâches, ce qui permet de précalculer les reformulations.

Nous fournissons des garanties sur la qualité des reformulations ainsi obtenues, montrant en particulier que toutes les bornes que l'on calcule en utilisant ces reformulations sont au moins aussi fortes que celles que l'on obtiendrait en faisant une relaxation préemptive du problème d'ordonnancement.

Cette technique permet de renforcer toutes les propagations de la contrainte cumulative reposant sur le calcul d'une borne énergétique, notamment l'Edge-Finding et le Raisonnement Énergétique.

Ce travail a été présenté lors de la conférence ROADEF 2014 [BB14] et a été publié dans le journal Discrete Applied Mathematics en 2017 [BB17a].

Raisonnement Énergétique en $O(n^2 \log n)$

Ce travail consiste en une amélioration de la complexité algorithmique de l'une de des propagations les plus puissantes pour la contrainte cumulative, le Raisonnement Énergétique de Erschler et Lopez, introduit en 1990.

Bien que cette propagation permette des déductions fortes, elle est rarement utilisée en pratique en raison de sa complexité cubique. De nombreuses approches ont été développées ces dernières années pour tenter malgré tout de la rendre utilisable (apprentissage automatique pour l'utiliser à bon escient, réduction du facteur constant de sa complexité algorithmique, etc).

Nous proposons un algorithme qui calcule cette propagation avec une complexité $O(n^2 \log n)$, ce qui constitue une amélioration significative de cet algorithme connu depuis plus de 25 ans. Cette nouvelle approche repose sur de nouvelles propriétés de la contrainte cumulative et sur une étude géométrique.

Ce travail a été publié sous une forme préliminaire lors de la conférence CP 2014 [Bon14] puis a fait l'objet d'une publication [Bon16] lors de la conférence ROADEF 2016, récompensée par la 2$^{\text{ème}}$ place au Prix du Jeune Chercheur.

université PARIS-SACLAY

ÉCOLE DOCTORALE
Sciences et technologies de l'information et de la communication (STIC)

Titre : Approches géométriques et duales en ordonnancement cumulatif

Mots-clés : optimisation, ordonnancement, programmation par contraintes, contrainte cumulative, raisonnement énergétique, dualité

Résumé : Ce travail s'inscrit dans le domaine de l'ordonnancement à base de programmation par contraintes. Dans ce cadre, la contrainte de ressource la plus fréquemment rencontrée est la cumulative, qui permet de modéliser des processus se déroulant de manière parallèle.
Nous étudions dans cette thèse la contrainte cumulative en nous aidant d'outils rarement utilisés en programmation par contraintes (analyse polyédrale, dualité de la programmation linéaire, dualité de la géométrie projective) et proposons deux contributions pour le domaine.
Le renforcement cumulatif est un moyen de générer des contraintes cumulatives redondantes plus serrées, de manière analogue à la génération de coupes en programmation linéaire entière. Il s'agit ici de l'un des premiers exemples de contrainte globale redondante.
Le Raisonnement Énergétique est une propagation extrêmement puissante pour la contrainte cumulative, avec jusque-là une complexité élevée en $O(n^3)$. Nous proposons un algorithme qui calcule cette propagation avec une complexité $O(n^2 \log n)$, ce qui constitue une amélioration significative de cet algorithme connu depuis plus de 25 ans.

Title: Geometric and Dual Approaches to Cumulative Scheduling

Keywords: optimization, scheduling, constraint programming, cumulative constraint, energy reasoning, duality

Abstract: This work falls in the scope of constraint-based scheduling. In this framework, the most frequently encountered resource constraint is the cumulative, which enables the modeling of parallel processes.
In this thesis, we study the cumulative constraint with the help of tools rarely used in constraint programming (polyhedral analysis, linear programming duality, projective geometry duality) and propose two contributions for the domain.
Cumulative strengthening is a means of generating tighter redundant cumulative constraints, analogous to the generation of cuts in integer linear programming. This is one of the first examples of a redundant global constraint.
Energy Reasoning is an extremely powerful propagation for cumulative constraint, with hitherto a high complexity of $O(n^3)$. We propose an algorithm that computes this propagation with a $O(n^2 \log n)$ complexity, which is a significant improvement of this algorithm known for more than 25 years.