Approximability of Precedence Constrained and Robust Scheduling Problems

Doctoral Dissertation submitted to the
Faculty of Informatics of the *Università della Svizzera Italiana*
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy

presented by
Nikolaus Mutsanas

under the supervision of
Prof. Luca Maria Gambardella
co-supervised by
Prof. Monaldo Mastrolilli

March 2010

Dissertation Committee

Prof. Antonio Carzaniga	Universitá della Svizzera Italiana, Switzerland
Prof. Evanthia Papadopoulou	Universitá della Svizzera Italiana, Switzerland
Prof. Friedrich Eisenbrand	Ecole Polytechnique Fédérale de Lausanne, Switzerland

Dissertation accepted on 17 March 2010

Prof. Luca Maria Gambardella
Research Advisor
Istituto Dalle Molle di Studi sull'Intelligenza Artificiale (IDSIA)

Prof. Monaldo Mastrolilli
Research Co-Advisor
Istituto Dalle Molle di Studi sull'Intelligenza Artificiale (IDSIA)

Prof. Michele Lanza
PhD Program Director

I certify that except where due acknowledgement has been given, the work presented in this thesis is that of the author alone; the work has not been submitted previously, in whole or in part, to qualify for any other academic award; and the content of the thesis is the result of work which has been carried out since the official commencement date of the approved research program.

Nikolaus Mutsanas
Lugano, 17 March 2010

Theoretical Computer Science is about showing that problems that **do not** exist **cannot** be solved assuming a conjecture **nobody** can prove.

Fred

Abstract

This thesis studies the approximability of scheduling problems in different contexts. Chapter 2 gives a short introduction to the field of scheduling theory and presents a simple single machine scheduling problem that will form the base for all variants considered in the remainder of this thesis. We point out that this scheduling problem, though long known to be efficiently solvable in its original form, becomes very interesting when additional restrictions are imposed. The restrictions in the focus of this thesis are precedence constraints among the jobs that need to be fulfilled by any feasible solution, or incomplete knowledge about the instance at hand.

We first study the precedence constraint version of the above mentioned single machine scheduling problem. This problem was first studied in the seventies, and still poses perplexing questions to researchers, concerning its approximability. Throughout the years several 2-approximation algorithms have been developed for it, with some special cases of precedence constraints allowing for better than 2 approximations. It was recently shown that this scheduling problem is strongly related to the VERTEX COVER problem of an appropriately defined graph. We establish a connection between this graph and a well-known graph in Dimension Theory of partial orders. We also extend a technique developed by Dorit Hochbaum that yields "good" approximation algorithms for the INDEPENDENT SET problem in graphs to the case that the so-called *fractional chromatic number* of the graph is bounded. Using the connections to dimension theory, we devise an algorithmic framework that unifies and often improves on the best known approximation algorithms for all previously considered special cases of partial orders. Besides its success in devising approximation algorithms for special cases of precedence constraints, the above sketched connection is also interesting in its own right. As an example, the polynomial solvability of 2-dimensional precedence constraints can be explained by the fact that the resulting graph becomes bipartite and vertex cover is known to be polynomially solvable for bipartite graphs.

We then study the above scheduling problems in the presence of uncertainty.

We show that this problem cannot be approximated within a logarithmic factor whenever the number of different scenarios, i.e. different configurations of the numerical parameters is unbounded. This result contrasts the difficulty in proving inapproximability results for the classical, non-robust problem and hints at the increase in complexity caused by the uncertain environment. We find it therefore surprising that we were able to develop a polynomial time 2-approximation algorithm for the case when only one of the two parameters is affected by uncertainty. The fact that our result holds in the presence of precedence constraints implies that it cannot be improved without improving upon the 2-approximation algorithm for the classical precedence constrained scheduling problem, a long standing open problem in scheduling theory. We further prove inapproximability results for the unweighted case and give a polynomial time algorithm for the case when both the number of scenarios and processing times / weights are bounded by some constant.

Parts of the work presented in Chapter 3 has been published as [AMMS07], [AMMS08] and [AMMS09]. Work presented in Chapter 4 has been published as [MMS08]. Research conducted was supported by Swiss National Science Foundation project 200021- 104017/1, “Power Aware Computing”, by the Swiss National Science Foundation project 200020-109854, “Approximation Algorithms for Machine scheduling Through Theory and Experiments II”, and by the Swiss National Science Foundation project PBTI2-120966, “Scheduling with Precedence Constraints”.

Acknowledgements

I would like to express my gratitude to

- my doctoral father, Monaldo, whose patience only admits lower bounds and who has become a paradigm of a researcher and teacher to me.
- Ola, my doctoral brother, friend, colleague, co-author, neighbor and soccer team-mate who always has some tricks up his sleeve.
- my colleagues and co-authors, Arash, Christoph, Giorgos, Monaldo and Ola who have taught me everything I know.
- our director, Luca, for supporting my studies and giving me the freedom to work on subjects of my interest.
- Prof. Carzaniga, Prof. Eisenbrand and Prof. Papadopoulou for offering to evaluate this dissertation and provide their valuable feedback.
- my colleagues and friends at IDSIA and USI that have made the stay at this institute a memorable time.
- Alexander, Carlo, Luca, Monaldo and Tom for trusting me with the task of assisting their excellent courses, from which I have learned a lot.
- *Orchestra Arcadia*, *Orchestra da Camera di Lugano*, *Trio di Ravecchia* and *Duetto αlphaβeto* for adding the right tone to my free time.
- my flatmates, neighbors and friends for making Castalia such a great place to live.
- the "Ticiniotes" Antonella, Chrysa, Giannis, Giorgos, Giorgos (2), Fotis and Ljuba for curing occasional home-sickness with a lot of greek jokes.
- my girlfriend, Ania, and my family, whose love and support was never affected by uncertainty.

Contents

Figures

Tables

Chapter 1

Introduction

Many commonly used techniques for solving numerical problems, simulating physical or social processes and manipulating information would be impossible today without the introduction of the fast computers. The insight that the limits of human computers have been long surpassed by the technological and scientific needs of our time has led people to wonder about the limits of machine computation.

1.1 Algorithms

The problems handled by a digital computer are required to be well-defined and formulated in a structural way, in the form of an *algorithm*. The mathematical dictionary of "MathWorld" gives the following definition [Wei]:

Definition 1.1.1 *An* algorithm *is a specific set of instructions for carrying out a procedure or solving a problem, usually with the requirement that the procedure terminate at some point.*

In other words, an algorithm is a rigid procedure that is defined with sufficient precision for a machine to be trusted with its execution. But the history of algorithms starts long before the invention of digital computers. All ancient civilizations had developed algorithms used in computations involved in construction, time measurement and other tasks, a prominent example being the Euclidean Algorithm developed as early as 375 BC. The construction of computers, however, lead to a previously unimaginable boost in the speed with which algorithms could be executed, and led people to wonder if all formally defined tasks can be solved by a computer. By brilliant arguments, Alan Turing [Tur36]

showed that there are well-defined mathematical problems for which no algorithm can exist that correctly solves all instances, the most famous example being the *halting problem* (see [Tur36]).

However, even the restriction to *decidable* problems, i.e. problems for which in principle there is an algorithm that would correctly solve any of its instances, poses big challenges to researchers. This is because an algorithm's requirements of computational resources such as time and memory might be so excessive that it becomes practically insignificant. The need to analyse algorithms in terms of the amount of resources needed for their execution has led to the field of Computational Complexity.

1.2 Computational Complexity and P vs. NP

It is obvious that the amount of resources needed by an algorithm to solve an instance of a problem grows with the size of the instance. The *efficiency* of an algorithm is quantified by studying how its number of basic operations scales as the size of the input is increased. The motivation for this measure comes from the fact that the efficiency of an algorithm is to a considerable extent much more important than the technology used to execute it [AB09]. As an example, comparing the grade-school algorithm for the multiplication of integers to a naive repeated addition algorithm, it is estimated that for 15-digit numbers, a fifth grader with pen and paper executing the former would outperform a modern supercomputer executing the latter.[1]

The above raised the question for a characterization of the efficiency of algorithms. In his celebrated paper [Edm65], Edmonds introduced the concept of *polynomial-time algorithm*, an algorithm whose runtime is bounded by a polynomial in the size of the instance. It is today generally agreed that an algorithm is efficient if it is a polynomial-time algorithm. One might argue that this definition has the flow that an algorithm which requires n^{100} operations is practical only for the smallest of instance sizes. However, support for this definition comes from our experience that almost all efficient algorithms run in time that is bounded by a polynomial of small degree.

A *complexity class* is a set of problems that can be solved within given resource bounds. In this context, the class P is defined as the set of problems that admit a polynomial-time algorithm. As discussed above, there is a consensus

[1] The number of basic operations — i.e. additions and multiplications of digits — needed by the grade-school multiplication of n-digit numbers is at most $2n^2$, while for the repeated addition it is at least $n10^{n-1}$.

that the class P contains those problems that can be solved efficiently. Naturally, a lot of effort has been made to place combinatorial problems in P, by devising polynomial-time algorithms that solve them. A very fruitful technique was provided by the *polynomial reducibility* among problems. A *polynomial reduction* from problem Π_1 to Π_2 is an algorithm that for any instance of Π_1 constructs, in polynomial time, an instance of Π_2 such that solutions to Π_2 correspond to solutions to Π_1. Thus, given a polynomial time algorithm for a problem Π_2, one can devise a polynomial time algorithm for Π_1 by polynomially reducing it to Π_2.

Nevertheless, for many basic combinatorial problems, such algorithms remain unknown, despite extensive efforts by many bright researchers. Our inability to devise efficient algorithms for this prominent group of problems has led to the development of a beautiful theory that unifies these failures into a deep mathematical conjecture, often cited as "P≠NP". The class NP contains the problems for which the validity of a suggested solution can be checked in polynomial time. In other words, it is different to the class P in that the "creative effort" of constructing a solution is no longer required, but instead an efficient way of validating solutions constructed externally is sufficient.

It is widely believed that the classes P and NP are different [Gas02], i.e. that removing the requirement for creative effort leads to a significantly different class of problems. This complies with our intuition that the inability to solve a difficult mathematical problem does not imply a difficulty in understanding the solution, once presented by the instructor. However, a proof of the statement P≠NP is inherently difficult: one must choose a problem and prove a claim about *all possible algorithms* that solve this problem, namely that they have a superpolynomial runtime. Due to this difficulty, the problem "P≠NP" remains a conjecture and has become one of the most fundamental open problems of our times (see e.g. [Cla00]).

Despite our inability to precisely classify problems according to their computational complexity, a lot of progress has been made in *interrelating* the complexities of different combinatorial problems. The class *NP-complete* is defined as the subclass of NP containing those problems $\Pi \in \text{NP}$ that have the property that all other problems in NP can be polynomially reduced to Π. In his seminal paper, Stephen Cook [Coo71] (and, independently, Leonid Levin [Lev73]) proved in the seventies that the problem of deciding the satisfiability of boolean formulas (SAT) is NP-complete. The importance of this proof is that it acted as a seed, based on which many other problems could be shown to also be NP-complete, by constructing polynomial reductions from these problems to SAT. This was demonstrated by Richard Karp who gave a list of 21 NP-complete problems us-

ing the hardness of SAT and polynomial reductions. Many more problems have been added to this class throughout the years (for a list of more than 200 basic NP-complete problems, see the online compendium [CK98]). By definition of the class NP-complete, these problems form a web of interconnected problems, such that a polynomial time algorithm for any of them would translate into a polynomial time for all other problems of the class as well. That is, the problems in the class NP-complete can be seen as different flavours of one, very hard, fundamental problem.

A problem is called *NP-hard* if it is at least as hard as the hardest problems in NP. More precisely, a problem Π is NP-hard if there exists an NP-complete problem that is polynomially reducible to Π. In this definition, by convention the length of some reasonable binary encoding of the instance is considered as the input size. A problem that remains NP-hard when the encoding of the instance is unary is called *strongly NP-hard*.

We point out a trivial "antisymmetry" in the inheritance of computational complexity. Given two problems Π_1 and Π_2, we say that Π_1 is a special case of Π_2 if all valid instances of Π_1 are also valid instances for Π_2. Naturally, an efficient algorithm for Π_2 implies an efficient algorithm for Π_1, since only a subset of instances is considered. Conversely, showing that Π_1 is NP-hard implies the same for the more general problem Π_2. Thus, the NP-completeness for problems in NP can often be established easily by identifying an NP-complete subproblem (see Problem 1.3.2 for an example).

1.2.1 Combinatorial Optimization

Similar to the problem of satisfiability, the beginning of the theory of NP-completeness revolved around *decision problems* i.e. problems that can be expressed as the computation of a boolean function. However, many combinatorial problems arising in practice are *optimization problems*: given an instance of a problem, the question is to find a solution, among a set of implicitly defined *feasible solutions* to this problem, such that a given measure on the quality of a solution is optimized. Seemingly easier versions of such problems are given by requiring only the *value* of an optimal solution (*evaluation problem*) or recognizing if there exists a solution of at least / at most a given value (*recognition problem*). However, it is easy to see that evaluation problems are no harder than recognition versions of the same problem. This is because a given hypothetical polynomial time algorithm that correctly answers the recognition problem can be combined with binary search in order to determine the exact value of the optimum. Furthermore, even though there is no general way of reducing optimization problems

to their evaluation versions, most problems exhibit the *self-reducibility* property (see Section 1.4.1), with which it can be shown that their optimization version is no harder than their evaluation version. This will in particular be the case for all problems considered in this thesis. For such problems, the three different versions mentioned above are essentially the same, in terms of complexity.

1.3 Approximation Algorithms

Many optimization problems that arise in practice are NP-complete. Assuming that $P \neq NP$, this implies that one cannot hope for a polynomial time algorithm that correctly solves all of the problem's instances. Thus, one must settle for a less ambitious goal, when dealing with such a problem. More precisely, one needs to sacrifice one of the following desired features of the sought algorithm:

- Solve the problem to optimality.
- Solve the problem efficiently.
- Solve arbitrary instances of the problem.

The field of Approximation Algorithms studies the design of algorithms that drop the first of those requirements, i.e. they solve arbitrary instances of a problem efficiently, though possibly suboptimally. Moreover, an approximation algorithm comes with a *guaranteed bound* on the deviation from optimality of the solution produced. A formal definition follows.

Definition 1.3.1 (Approximation Algorithm) *Let Π be a minimization (respectively, maximization) problem. Let $\varepsilon > 0$ and set $\rho = 1+\varepsilon$ (respectively, $\rho = 1-\varepsilon$). An algorithm A is called a* ρ-approximation algorithm *for the problem Π, if for all instances I of Π it computes a feasible solution with the objective value $A(I)$ such that*

$$|A(I) - OPT(I)| \leq \varepsilon \cdot OPT(I)$$

where $OPT(I)$ denotes the value of the optimal solution to the instance I. The value ρ is called the performance guarantee *or the* worst case ratio *of the approximation algorithm A.*

This definition seems counterintuitive at first sight, since it requires a guarantee involving the optimal solution of the problem. However, as discussed in Section 1.2, solving the evaluation version of a problem is computationally not

easier than solving the optimization version. This observation hints to an important part in the design of approximation algorithms: bounding the value of an optimal solution and analyzing the worst-case approximation ratio. We will illustrate these concepts with an example of a 2-approximation algorithm for the LOAD BALANCING problem.

Problem 1.3.2 (LOAD BALANCING)

Given: *A set of m identical* machines*, a set of n* jobs $N = \{j_1, j_2, \ldots, j_n\}$ *and for each job j_i a* processing time $p_i \in \mathbb{Q}$.

Find: *A partitioning of N into m partitions $N_1, \ldots, N_m$ such that the jobs in N_i are scheduled on machine i in an arbitrary order and without gaps, such that the maximum load among all machines (i.e. the* makespan*)*

$$\max_{1 \leq i \leq m} L_i$$

is minimized, where the load of machine i is defined by $L_i := \sum_{j \in N_i} p_j$.

It is easy to see that this problem is NP-hard, since it contains the problem 2-PARTITION as a special case. The following algorithm, due to Graham [Gra66], is one of the first approximation algorithms and achieves an approximation ratio of 2: *arbitrarily order the jobs and schedule them in this order, each time choosing the currently least loaded machine*.

This algorithm clearly returns a feasible solution. In order to prove the approximation ratio of 2, we need to lower bound the value of an optimal solution, as discussed above.

Let L^* be the optimal makespan. Since all jobs need to be scheduled at some point, we know that there will be a machine with load at least $p_{\max} := \max_{1 \leq j \leq n} p_j$ and thus $p_{\max} \leq L^*$. Moreover, in the best case it will be possible to distribute the total processing time of all the jobs evenly across machines, which gives another lower bound on L^*, namely $\frac{1}{m} \sum_{j=1}^{n} p_j \leq L^*$.

Now the analysis of the approximation ratio can be done as follows. Let k be the most loaded machine in solution returned by the approximation algorithm and let j be the last job scheduled on this machine. When j was assigned, k was the least loaded machine, i.e. all machines have load at least $L_k - p_j$. In other words,

$$L_i \geq L_k - p_j, \ \forall 1 \leq i \leq m.$$

If we sum over all machines and divide by m we get

$$L_k - p_j \leq \frac{1}{m} \sum_{i=1}^{m} L_i = \frac{1}{m} \sum_{j=1}^{n} p_j \leq L^*.$$

Now we only need to rewrite the load of machine k as

$$L_k = \underbrace{L_k - p_j}_{\leq L^*} + \underbrace{p_j}_{\leq p_{\max} \leq L^*} \leq 2L^*.$$

This means that the algorithm returns a feasible solution that has makespan at most twice the optimal makespan, and is thus a 2-approximation.

The above example shows the importance of devising lower bounds, in the design of approximation algorithms. A particularly successful method of bounding optimal solution values has been provided by the theory of *linear programming*. Many problems allow for a natural, exact formulation by an *Integer Linear Program (ILP)*. Such a formulation uses binary decision variables, linear constraints on these variables and a linear objective function. Since there are NP-complete problems that admit such an exact formulation, solving an ILP is NP-complete as well. However, the situation changes when the integrality constraint on the variables is dropped. The resulting *linear programming (LP)* formulation can be solved efficiently to optimality, but the resulting solution is not feasible for the integral problem. Nonetheless, the optimal value thus achieved often gives a good bound on the actual optimum, and a wide range of algorithms has been devised that work by "rounding" the fractional solution to a feasible (integral) one, without deteriorating the objective value too much. The maximum ratio between an optimal fractional and an optimal integral solution for an LP is called the *integrality gap of the LP*. We defer a detailed example to Section 3.3 where we show a very simple 2-approximation algorithm for the VERTEX COVER problem, based on LP-rounding.

1.3.1 Approximation Schemes

For some problems, the development of an approximation algorithm can be parametrized in such a way that a "tuning" of the trade-off between runtime and approximation guarantee is possible. Instead of an algorithm, such results yield an algorithmic *scheme*, defined below.

Definition 1.3.3 *Let* Π *be a minimization problem (respectively, maximization) problem.*

- *An* approximation scheme *for problem* Π *is a family of* $(1+\varepsilon)$*-approximation algorithms* A_ε *(respectively,* $(1-\varepsilon)$*-approximation algorithms* A_ε*) for problem* Π *over all* $0 < \varepsilon < 1$.

- *A* polynomial time approximation scheme (PTAS) *for problem* Π *is an approximation scheme whose time complexity is polynomial in the input size.*
- *A* fully polynomial time approximation scheme (FPTAS) *for problem* Π *is an approximation scheme whose time complexity is polynomial in the input size and also polynomial in* $1/\varepsilon$.

For many problems a PTAS is the best one can hope for. Indeed, the following theorem states that, under certain conditions, only so-called *weakly NP-complete* problems may admit an FPTAS (see e.g. [Vaz01]).

Theorem 1.3.4 (Strongly NP-hard and "natural" ⇒ No FPTAS)
Let Π *be a strongly NP-hard integer-valued optimization problem such that on any instance* I *of* Π, $OPT(I) < p(|I_u|)$ *where* p *is some polynomial and* I_u *is a unary encoding of instance* I. *Then* Π *does not admit an FPTAS, assuming* P $\neq$ NP.

We point out that the conditions required by this theorem for the strongly NP-hard problem are naturally fulfilled by most problems. Thus, unless otherwise specified, strong NP-hardness implies that the problem does not admit an FPTAS.

1.3.2 Inapproximability proofs

Every approximation algorithm achieving a ratio ρ naturally raises the question of whether this ratio can be improved or not. Inapproximability results are important tools that guide the search for the best possible efficient approximation algorithm for a given problem. The following technique, often referred to as "the gap technique" (see e.g. [SG76; GJ76; LR78a]) is one of the oldest methods to prove that a problem cannot be approximated to arbitrary precision in polynomial time.

Theorem 1.3.5 (The gap technique) *Let* Π_d *be an NP-hard decision problem, let* Π_m *be a minimization problem, and let* τ *be a polynomial time computable transformation from the set of instances of* Π_d *into the set of instances of* Π_m *that satisfies the following two conditions for fixed integers* $a < b$*:*

- *Every YES-instance of* Π_d *is mapped into an instance of* Π_m *with optimal objective value* at most a.
- *Every NO-instance of* Π_d *is mapped into an instance of* Π_m *with optimal objective value* at least b.

Then problem Π_m does not have a polynomial time ρ-approximation algorithm with $\rho < b/a$ unless P=NP.

Note that this theorem in particular states that problem Π_m does not admit a PTAS. Such types of reductions are often called "gap-introducing reductions". Furthermore, we note that, similar to polynomial reductions in order to establish the NP-completeness of a problem, approximation-preserving reductions can be used to make statements about the approximability of a problem (see e.g. the *L-reduction* [PY91]).

1.3.3 Approximation Gap

The two-fold nature of research conducted in the field of Approximation Algorithms led to the concept of the *approximation gap* of a problem. Intuitively, the approximation gap serves as an illustrative measure of our understanding of the approximability of a problem, by describing the discrepancy between the best known positive and negative results on this problem. More precisely, the approximation gap of a problem is the interval between the best known approximation algorithm and the strongest known inapproximability result for this problem.

Our understanding of the approximability of a problem is considered to be complete when the approximation gap vanishes. For instance, the approximability of the 3SAT problem is considered well-understood, since there exists a trivial 7/8-approximation algorithm[2], while it has been shown by Johan Håstad [Hås97] that an algorithm achieving an approximation ratio of $(7/8 - \varepsilon)$ cannot exist, assuming P $\neq$ NP. On the other hand, the VERTEX COVER problem mentioned earlier is one of the most prominent problems that retain a non-trivial approximation gap to this day. In the case of VERTEX COVER there are many simple 2-approximation algorithms (see e.g. Section 3.3.1), while the strongest inapproximability result, due to Dinur & Safra [DS02], states that the problem cannot be approximated within a factor of 1.3606.

1.4 Unique Games Conjecture

The *Unique Games Conjecture* is a complexity theoretic conjecture proposed by Subhash Khot [Kho02] in 2002. Before discussing its implications, we give a

[2]This can be achieved by derandomizing a simple coin-flipping algorithm that randomly chooses a truth value for each variable without looking at the instance (similar to the one presented in Section 1.4.1).

definition of the problem.

Problem 1.4.1 (UNIQUE GAMES)

Given: *An undirected, connected graph $G = (V, E)$, a set of colors C and for each edge $\{i, j\}$, $i < j$, a permutation $\pi_{i,j} : C \to C$.*

Find: *A coloring of the graph that maximizes the number of satisfied edges, i.e. an assignment of colors to vertices $c : V \to C$ such that the number of edges $\{i, j\}$ for which $\pi_{i,j}(c(i)) = c(j)$ holds is maximized.*

The "uniqueness" in the name of the problem comes from the fact that the mapping associated with each edge $\{i, j\}$ is a permutation, and thus the color of i uniquely determines the color of j in any coloring satisfying all edges.

Note that deciding whether there exists a coloring satisfying all edges can be solved efficiently: start at any vertex of the graph and try all possible colors for this vertex, propagating the uniquely defined colors for adjacent edges until a valid coloring has been found or all initial colors lead to unsatisfied edges. However, the approximability of this problem exhibits a very different behaviour. Indeed, the Unique Games Conjecture (UGC) states that, for every constants $\varepsilon, \delta > 0$ there is a color set C of sufficiently big size that depends on ε, δ such that it is NP-hard to distinguish the following two cases for any unique game with color set C:

1. there is a coloring satisfying at least a fraction $(1 - \delta)$ of the edges.

2. any coloring satisfies at most a fraction ε of the edges.

A lot of effort has been invested into the settlement of this conjecture. Khot's suggestion that current algorithmic techniques seem unable to design such an algorithm is supported by the fact that it defied very intensive attempts towards its disproof. Indeed, our current understanding locates the conjecture on the fine line between being true and being false [AB09].

What makes the Unique Games Conjecture so interesting is that it acts as a seed for stronger inapproximability results, in the same way that the NP-completeness of SAT acts as a seed for intractability results. As an example, the trivial 2-approximation algorithm for the VERTEX COVER problem, presented in Section 3.3.1 is proven to be best possible, if one is willing to assume the Unique Games Conjecture [KR08]. Our inapproximability result for the robust scheduling problem discussed in Section 4.4.2 improved from 6/5 to 4/3 based on the Unique Games Conjecture. As more and more inapproximability results are obtained assuming the UGC, its settlement becomes increasingly important

and would constitute a leap forward in our understanding of Approximation Algorithms.

As a side-note, there are also weaker assumptions than the Unique Games Conjecture that are still stronger than the traditional assumption P $\neq$ NP that are useful for inapproximability proofs. As an example, in Section 4.4.1 we show an inapproximability proof based on the assumption that the class NP does not have quasi-polynomial algorithms, i.e. there is no algorithm that can solve all NP-hard problems with an asymptotic runtime of $O(2^{\text{poly}(\log n)})$, where n is the size of the input. This assumption, while stronger than P $\neq$ NP, is still widely believed to be true.

1.4.1 Randomization and Approximation

Randomization has proven to be a very useful tool in the design of approximation algorithms, and we will use it extensively in this thesis. A *randomized* or *probabilistic algorithm* is an algorithm that has access to a source of usually uniformly distributed random bits, often referred to as "coin-flips", which guide its execution. The performance of a randomized algorithm is measured in terms of the *expected* quality of the solution.

A common method towards a guarantee for such algorithms despite randomness, is given by repetition: the probability that a randomized algorithm returns a solution at least as good as the expectation is increased by executing the algorithm several times and retaining the best solution among all trials. However, such an algorithm still remains probabilistic.[3] Nonetheless, in some cases a purely deterministic can be devised that matches the guarantee given in expectation by the randomized algorithm, using *derandomization*. In particular, this will be the case for almost all algorithms presented in Chapter 3. We briefly sketch a common method for derandomization, the *method of conditional probabilities*.

The method of conditional probabilities

Many combinatorial problems exhibit the so-called *self-reducibility* property. This property, besides being useful in finding a solution given an oracle for the decision version, is helpful for the derandomization of algorithms. Intuitively, the self-reducibility property implies that completing a given partial solution for an instance is again another (smaller) instance of the same problem. We demonstrate this property on the example of MAX-2-SAT, and devise a deterministic

[3] For a comment on the statement "with high probability", see [Pra07].

3/4-approximation algorithm using derandomization. First, we give a definition of the problem:

Problem 1.4.2 (MAX-2-SAT)

Given: *A boolean formula ϕ in conjunctive normal form with n variables and m clauses, such that each clause contains exactly two literals (i.e. variables or their negations).*

Find: *A truth assignment for all variables that maximizes the number of satisfied clauses, i.e. the clauses such that at least one of their literals is true.*

The following is a very simple randomized algorithm for this problem: for each of the variables, flip a coin and set its truth value according to the outcome of the coin flip, e.g. set it to true if heads. The expected performance of this algorithm can be computed as follows. For each clause, say $(x_1 \vee \overline{x_2})$, there are three truth assignments that satisfy it (namely $(0,0),(1,0),(1,1)$) and one that does not (namely $(0,1)$). Thus, the probability of satisfying any clause by random truth assignment is $3/4$ and the expected number of satisfied clauses is $3m/4$. As m is a trivial upper bound on the value of an optimal solution, the above method yields a randomized 3/4-approximation algorithm.

The above algorithm can be derandomized as follows. Consider an arbitrary ordering of the variables $x_1, x_2, \ldots, x_n$ of ϕ, and consider a binary tree of depth n whose leaves are all binary strings of length n (See Figure 1.1 for an example). This tree can be interpreted as the computation tree of all truth assignments of ϕ. A node at depth $1 < i < n$ represents an instance in which the first $n - i$ variables have been assigned some truth value. The self-reducibility says that each node v corresponds to a Max-2-Sat instance ϕ_v such that the maximum number of satisfied clauses of ϕ in the assignments (leafs) of the subtree rooted at v is given by the number of clauses satisfied by the partial truth assignment plus the maximum number of satisfiable clauses in ϕ_v. This means that the *expected* number of satisfiable clauses in any subtree can be computed in polynomial time. A deterministic algorithm with a guarantee at least as good as the average is now obvious: start at the root of the tree and compute the expected number of satisfied clauses for each of the two subtrees. Assign the truth-value corresponding to the subtree with the highest expected value and iterate until a leaf node (i.e. a truth assignment) is reached.

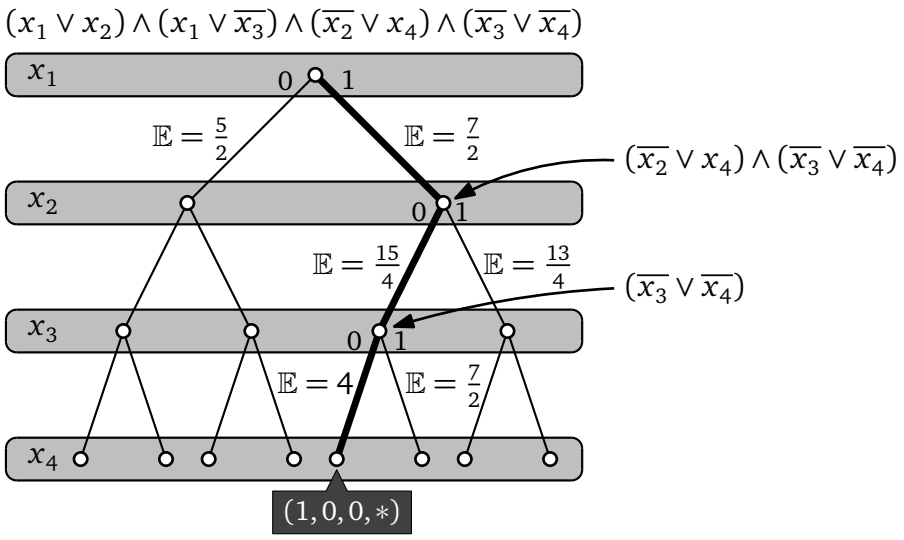


Figure 1.1. An example of the derandomization procedure for MAX-2-SAT. Notice that the resulting deterministic algorithm returns a solution that satisfies 4 clauses, i.e. at least as many as the randomized algorithm in expectation, i.e. $4 \cdot 3/4 = 3$.

1.5 Robust Optimization

The field of Robust Optimization [KY97] was created by the need to consider combinatorial problems that are defined in uncertain environments. This uncertainty naturally presents additional difficulties, since a guarantee is required for a solution of an instance that is not fully known. Robust Optimization sets as a goal to optimize the worst-case scenario over a set of possible realizations of the numerical parameters. That is, a solution is evaluated in a range of possible configurations for the numerical parameters, and its worst-case performance is taken as a measure for its quality in the uncertain environment.

The emphasis in robust optimization lies in the "worst-case" objective. The desired solution will in general perform suboptimally in the actual scenario, but this is traded off for acceptable performance in all scenarios. Thus, the field of robust optimization becomes interesting when a rigid guarantee is necessary. As an example where this principle is applied, many travelers choose to take the train to the airport when they are about to take a flight, even though driving by car is (in most cases) faster. The importance of not missing a flight makes the suboptimal in duration train trip more attractive than the car ride, whose duration is uncertain due to traffic.

The computational complexity of robust optimization problems are in general higher than their classical, non-robust counterparts. Since the classical problems can be viewed as robust problems with a single scenario, any hardness results for the classical problems are inherited by their robust versions. Moreover, the definition of different scenarios allows for the "overlay" of several instances of a problem into one, robust instance, leading to more complex problems. We demonstrate this phenomenon in Section 4.4.1, where we employ scenarios in order to present a reduction from the LABEL COVER problem to a robust scheduling problem.

1.6 Terminology and notation

The following graph theoretic concepts are going to be used throughout the thesis (see Figure for an example):

A *graph* G is a mathematical structure consisting of a set V (often assumed to be equal to the set $\{1, 2, \ldots, n\}$ of n *vertices* and a set E of m *edges*. In case the graph is *undirected*, edges consist of a set of unordered pairs of V, i.e. of the type $\{i, j\}$ for $i, j \in V$, while for *directed* graphs they are ordered pairs of the type (i, j) for $i, j \in V$. We denote such a graph by $G(V, E)$. We say that i and j are *adjacent* when the edge $\{i, j\}$ is part of the graph G. Moreover, we say that the edge $\{i, j\}$ is *incident* to both of the vertices i and j, which are also called its *endpoints*. A set of vertices $S \subseteq V$ such that no two of them are adjacent is called *vertices!independent*. We say that a set of vertices $V' \subseteq V$ *covers* a set of edges $E' \subseteq E$, if for each edge $e \in E'$ at least one of its endpoints is in V'. We say that a set $V' \subseteq V$ *induces* an edge $e \in E$ when both endpoints of e are contained in V'. A graph $G'(V', E')$ is called an *induced subgraph* of the graph $G(V, E)$ if $V' \subseteq V$ and the edge set E' contains the edges of E that are induced by V'

The number of edges that are incident to a given vertex i is called the *degree of* i, denoted by $\deg(i)$. The maximum degree among all vertices of a graph G is called the *degree of* G. A graph is called *bipartite* if there exists a 2-partitioning of its vertices — i.e. a division of the set of vertices into two sets V and W such that each vertex is in exactly one of V, W — such that neither V nor W induce any edges. A graph $G(V, E)$ is called a *clique* or *complete graph* when any two vertices in V are adjacent, i.e. $E = \binom{V}{2}$.

Note that the above definitions easily generalize to *hypergraphs*, i.e. graphs whose edges may have cardinality greater than two. A hypergraph with hyperedges of the same cardinality k is called a *k-regular hypergraph*.

For the analysis of asymptotic runtimes, approximation ratios or instance

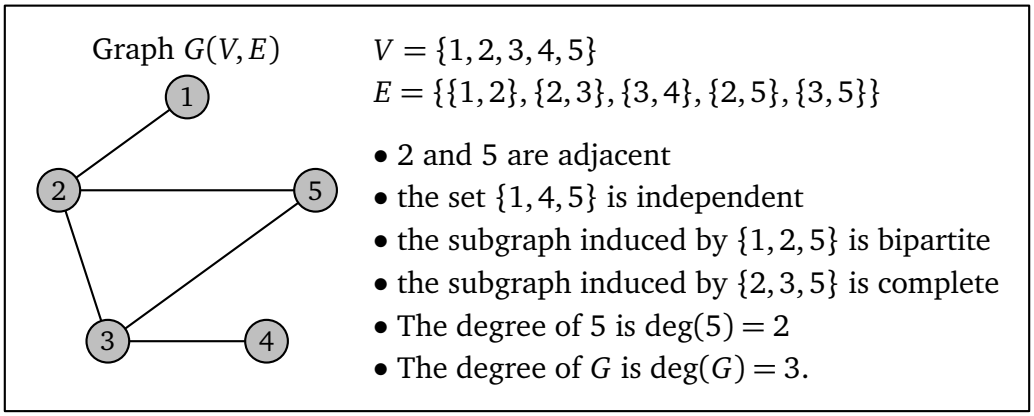


Figure 1.2. Examples of the graph theoretic definitions used in this thesis.

sizes, we use the *Big-Oh notation* (see e.g. Section 0.3 in [AB09]).

1.7 How to read this thesis

Chapter 2 gives a short introduction to scheduling and presents a simple, efficiently solvable scheduling problem. Intractable variants of this problem and their approximability form the content of subsequent chapters. In Chapter 3 we consider the classical problem in which precedence constraints are added, while in Chapter 4 robust variants of this problem are studied. Both chapters are largely self-contained, with references to Chapter 3 being made whenever relevant results are presented in Chapter 4.

Both Chapters 3 and 4 give an overview of the relevant literature before presenting our contributions. In Chapter 3 we present an algorithmic approximation framework consisting of several components. We sketch this framework in the beginning before going into the details, and revisit the framework in the end of the chapter. This sketch serves as a roadmap to the remainder of the chapter, and the reader is advised to use it as a reference.

Finally, Appendix A contains an index of the combinatorial problems that appear in this thesis for easy reference.

Chapter 2

Scheduling Theory

One of the earliest studied fields in Combinatorial Optimization is the field of Scheduling Theory. Scheduling is a decision-making process that plays an important role in most manufacturing and service industries [PC99]. It naturally arises whenever a limited number of resources need to be allocated to the processing of tasks.

What constitutes a *resource* may range from a processing unit of a computer or a machine in an assembly line to a group of employees of a company, or a platform in a train station. Similarly, a *task* may be a computer program, a stage in the production of an item, an order to be processed by a company or an arrival and departure of a train. The objectives to be optimized can also take many forms, such as minimizing the time spent until the last task completes or minimizing a more sophisticated function that takes into account priorities of different tasks.

2.1 A simple scheduling problem

Consider the following example of a scheduling problem:

Example 2.1.1 [Bakery Scheduling] A bakery has received a set of orders from different customers to be processed during the day. Each order needs a different amount of time to be processed, and is immediately delivered to the customer as soon as it is completed. The bakery would like to process the orders in such a way that the waiting time of the customers is as small as possible. However, the bakery would also like to favor orders given by their regular customers, by valuing their waiting times higher. Find a schedule that best meets the above stated goals.

Scheduling Theory deals with the *combinatorics* of scheduling problems. Mathematical models are used in order to abstract from the concrete interpretation of the tasks and resources and only consider its significant parameters. In the following, we will use the terms *job* and *machine* to denote tasks and resources respectively, regardless of their interpretation.

Let us try to formalize the scheduling problem of Example 2.1.1 in order to put it into this context. We define a job j_i for each of the orders taken by the bakery. We call the amount of time needed to process the i-th order the *processing time of job* j_i and represent it by a non-negative rational $p_i \in \mathbb{Q}$. In order to model the different priorities given to customers, we need to introduce a further parameter for each job, which we will call its *weight*, represented by a non-negative rational $w_i \in \mathbb{Q}$.

Clearly, if we ignored the fact that regular customers are favored, a natural objective would be to minimize the function $\sum_{i=1}^{n} C_i$, where C_i is the time at which job j_i completes in the schedule. Minimizing this sum simply means not to allow any gaps between the processing of orders. Thus, given a total ordering (or permutation) L of the jobs $\{j_1, j_2, \ldots, j_n\}$, the completion time of job C_i is

$$C_i = \sum_{\substack{k \in \{1,\ldots,n\} \\ k \leq i \text{ in } L}} p_k.$$

To model the priorities, we can simply assign a high weight to jobs corresponding to orders given by regular customers and a low weight to the remaining ones, and replace the objective by $\sum_{i=1}^{n} w_i C_i$. Note that, by allowing for any non-negative rational to be the value of a weight, the distinction between regular and occasional customers can be refined accordingly.

With the above, the scheduling problem described in Example 2.1.1 can be formally defined as follows.

Problem 2.1.2 (Single Machine Scheduling)

Given: *A set of* jobs $N = \{j_1, j_2, \ldots, j_n\}$ *and for each job* j_i *a processing time* $p_i \in \mathbb{Q}$ *and a weight* $w_i \in \mathbb{Q}$.

Find: *An ordering of the jobs* N *that minimizes the sum of weighted completion times*

$$\sum_{i=1}^{n} w_i C_i$$

where the completion time C_i *of job* j_i *is the time at which it completes in the schedule.*

Note that, since each ordering of the jobs is a feasible solution to the Single Machine Scheduling problem, the optimal solution can be achieved by a brute-force algorithm that systematically generates all $n!$ permutations keeping track of the currently best known ordering. However, the following observations show that this problem can be solved in a much more efficient way.

Assume that we are given an instance of Problem 2.1.2 in which all weights are equal to 1. Since the processing time of each job is included in the completion time of all jobs that follow, it is advantageous to schedule jobs with a small processing time first. Thus, scheduling the jobs in non-decreasing order of their processing time gives the optimal solution. Conversely, if we are given an instance in which all processing times are equal to 1, it is advantageous to schedule jobs with a small weight last, and the best solution would schedule the jobs in non-increasing order of their weights.

These observations easily carry over to the general case: when both the processing times and weights of the jobs are allowed to be arbitrary non-negative rationals, the best choice for the position of a job is given by the *trade-off* – i.e. the ratio – between its processing time and its weight. This intuitive claim is easy to prove and has been long known as *Smith's rule*. The ratio between the processing time and the weight of a job is often called the *density* of the job and denoted by $\rho_i = p_i/w_i$.

Theorem 2.1.3 (Smith's rule [Smi56], 1956)
Assume an instance of Problem 2.1.2. Scheduling the jobs in non-decreasing order of their densities, breaking ties arbitrarily, gives an optimal solution.

Proof. First, observe that the schedule thus constructed is unique up to reordering of jobs with equal density and that all these (equivalent) schedules have the same value. Now, assume, towards contradiction, that these schedules are suboptimal. Then, any optimal solution must violate Smith's rule, i.e. in any optimal solution L^*, there exists a pair of jobs such that the earlier scheduled among them has a strictly greater density than the other. In particular, there must be a pair of *consecutive* jobs, say j_k, j_l, such that $\rho_k > \rho_l$ (see Figure 2.1). However, it is easy to see that exchanging the order of j_k and j_l leads to a solution with a strictly better value, contradicting the optimality of L^*.

Indeed, let L' be the schedule obtained from L^* by exchanging the order of j_k and j_l (see Figure 2.1). Let $A \subseteq N$ be the indices of jobs proceeding $\{j_k, j_l\}$ and $B \subseteq N$ the indices of jobs following $\{j_k, j_l\}$ in both schedules L^* and L'. Observe that any job j_i with $i \in A \cup B$ contributes the same to the objective value of L^*

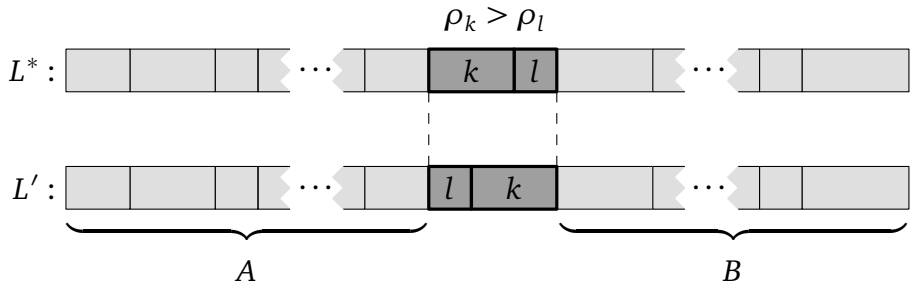


Figure 2.1. Exchanging the order of j_k and j_l does not affect the contributions of jobs in $A \cup B$ to the objective function.

and L'. Thus

$$\mathrm{val}(L^*) = \sum_{i=1}^{n} w_i C_i = \sum_{i \in A \cup B} w_i C_i + w_k C_k + w_l C_l$$

while

$$\mathrm{val}(L') = \sum_{i=1}^{n} w_i C'_i = \sum_{i \in A \cup B} w_i \underbrace{C'_i}_{=C_i} + w_k C'_k + w_l C'_l$$

where C_i and C'_i are the completion times of job j_i in L^* and L' respectively. By definition of the completion time, we have

$$\begin{array}{rclcrcl} C_k &=& \sum_{i \in A} p_i + p_k & & C_l &=& \sum_{i \in A} p_i + p_k + p_l \\ C'_k &=& \sum_{i \in A} p_i + p_k + p_l & & C'_l &=& \sum_{i \in A} p_i + p_l \end{array}$$

Thus,

$$\begin{array}{rcl} \mathrm{val}(L^*) - \mathrm{val}(L') &=& w_k C_k + w_l C_l - w_k C'_k - w_l C'_l \\ &=& w_k (C_k - C'_k) - w_l (C'_l - C_l) \\ &=& -w_k p_l + w_l p_k \\ &>& 0 \end{array}$$

where the last inequality follows from the assumption $p_k/w_k = \rho_k > \rho_l = p_l/w_l$. This contradicts the optimality of L^* and concludes the proof. □

As a corollary of Theorem 2.1.3, the Single Machine Scheduling problem admits a polynomial-time algorithm: all there is to do is compute the densities of the jobs and sort them non-decreasingly.

Corollary 2.1.4 Problem 2.1.2 can be solved in $O(n \log n)$ time.

The single machine scheduling problem is one of the earliest studied problems in Scheduling Theory and has been long known to be solvable efficiently. However, the situation changes dramatically when additional restrictions on the possible orderings are imposed, or when the exact numerical parameters of the instance are unknown. This is the content of the following chapters. In Chapter 3 we study the problem in which the instance description is extended to include a partial order that describes precedence constraints among the jobs. In Chapter 4 we study the problem in which the numerical parameters are uncertain.

2.2 Graham notation

The field of scheduling theory allows for a wide range of scheduling problems to be defined, by combining different machine environments with additional restrictions on the feasibility of a solution. Those can be further combined with several natural objective functions to produce a big number of interesting scheduling problems. The rich diversity of this field yielded the need for a standardized way of representing scheduling problems. To this goal, Graham, Lawler, Lenstra & Rinnooy Kan [GLLR79] introduced a concise way of representing scheduling problems[1] by defining three fields α, β and γ, often written in the form

$$\alpha|\beta|\gamma$$

each one defining one of the above mentioned parameters. More precisely:

α**:** *Machine environment*. We distinguish between two kinds of problems, in each of which α can assume several different values: in *single stage problems* each job consists of a single entity, while in *multi-stage problems* each job consists of several entities, called *operations*, that need to be processed in order to complete the job. Some values of the field α are given in Table 2.1.

β**:** *Job characteristics*. In this field, additional job characteristics or restrictions are noted. For example, the case in which all jobs have the same processing time p can be expressed by putting $p_i \equiv p$. In some contexts, jobs have an earliest and latest time in which they can be processed, called their *release time* r_i and *deadline* d_i respectively. If a job may be *preempted* and

[1] Nowadays often referred to as the *Graham Notation*

Single stage problems	
1	*Single machine* environment.
P	*Parallel machines* environment. There are m identical machines available for the processing of the jobs[1].
Q	*Uniform parallel machines* environment. There are m machines with different given speeds[1]. The processing time of each job is to be divided by the speed of the machine it is run on.
U	*Unrelated parallel machines* environment. There are m, each specifying the processing time for each of the jobs in case that job is run on it[1].
Multi-stage problems	
O	*Open Shop* environment. Each job contains one operation for each of the m machines. The order in which the operations are processed can be arbitrary.
F	*Job Shop* environment. Each job contains one operation for each of the m machines. The order in which the operations are processed is fixed for each job.
J	*Flow Shop* environment. Each job contains one operation for each of the m machines. The order in which the operations are processed is fixed and common to all jobs.

[1] Whenever the number of machines is a fixed constant, its value is appended to the environment notation

Table 2.1. Some of the possible values of the parameter α describing the machine environment of a scheduling problem

resumed at a later point, this is noted by "pmtn". The presence of precedence constraints among the jobs is denoted by "prec", or by the name of the partial order when the class of precedence constraints is restricted, e.g. "interval-orders".

γ: *Objective function*. Natural objective values usually depend on the following values: deadline d_i, *completion time* C_i, *lateness* $L_i = C_i - d_i$, *tardiness* $T_i = \max\{0, C_i - d_i\}$ to name a few. For example, a common objective to be minimized is the maximum value of the values of C_i, E_i, T_i, such as the *makespan* $C_{\max}$. Another common objective is to minimize the sum of these values, or the weighted version thereof when weights are present, such as the *weighted sum of completion times* $\sum w_i C_i$. The latter objective will be

the focus of this thesis.

As an example, Problem 2.1.2 from the previous Section is denoted by

$$1 \mid\mid \sum w_i C_i$$

in the Graham notation.

Chapter 3

Single Machine Scheduling with Precedence Constraints

In this chapter we study a classical problem in scheduling theory, denoted by $1|\text{prec}|\sum_j w_j C_j$ in Graham's notation (see Section 2.2). We show that this problem bears strong ties to the dimension theory of partial orders. Combining this connection with the fact that this problem is a special case of the vertex cover problem, we are able to devise an algorithmic framework that extends and unifies the currently best known approximation algorithms for all previously considered special cases of precedence constraints.

In Section 3.1 we define the problem formally and give an overview of the relevant literature. We choose to first give an intuitive sketch of our framework in Section 3.2, with the individual components being discussed in the subsequent sections. In Section 3.3 we prepare the ground for the design of our framework by reviewing the connection between $1|\text{prec}|\sum_j w_j C_j$ and the vertex cover problem. The connection to the dimension theory of partial orders is studied in Section 3.4. In Section 3.5 we put everything together to derive the algorithmic framework that yields "good" approximation algorithms whenever the precedence constraints are of "low complexity". We apply this framework in Section 3.6 and present several classes of precedence constraints that allow for a better approximation than the general case. Finally, some directions for future research are suggested in Section 3.7.

3.1 Introduction

We consider the following classical scheduling problem:

Problem 3.1.1 ($1|\text{prec}|\sum_j w_j C_j$)

Given: *A set $N = \{j_1, j_2, \ldots, j_n\}$ of n jobs, for each job j_i a processing time $p_i \in \mathbb{Q}$ and a weight $w_i \in \mathbb{Q}$, and a* partial order $\mathbf{P}(N, P)$ *defined on the set of jobs N, i.e. a reflexive*[1]*, antisymmetric and transitive relation P on N. If $(i, j) \in P$, i needs to have completed before j can be processed.*

Find: *A schedule of N without interruptions, i.e. a total ordering L of the jobs on a single machine that respects the precedence constraints and minimizes the weighted sum of completion times $\sum_{i \in N} w_i C_i$.*

We illustrate the above definition in terms of the following example instance. Let $N = \{j_1, j_2, j_3, j_4\}$ be the set of jobs and let $p = (p_1, p_2, p_3, p_4) = (3, 3, 2, 4)$ and $w = (w_1, w_2, w_3, w_4) = (1, 4, 2, 5)$ be the vectors defining the processing times and weights of the jobs, respectively. Finally, let $\mathbf{P}(\{(j_1, j_2), (j_3, j_4)\}, N)$ be the partial order describing the precedence constraints. This instance is depicted in Figure 3.1, with rectangles representing the jobs, subscripts representing the weights of the jobs and the processing times encoded as the lengths of the rectangles. An arrow connecting two jobs $i \to j$ represents a precedence constraint of the form $(i, j) \in P$. Note that scheduling the jobs according to their density, as suggested by Smith's rule, gives the *infeasible* schedule (j_2, j_4, j_3, j_1) (schedule (a) in Figure 3.1). This schedule is infeasible because it does not comply with the precedence constraints given by the partial order: j_3 is to be scheduled after j_1, whereas the schedule puts these two jobs in the reverse order. On the other hand, the schedule $L = (j_2, j_4, j_1, j_3)$ is a *feasible* schedule, since it respects all precedence constraints given by the partial order. Its objective value is

$$\begin{aligned} \text{val}(L) &= \sum_{i=1}^{4} w_i C_i \\ &= w_2 p_2 + w_4(p_2 + p_4) + w_1(p_2 + p_4 + p_1) + w_3(p_2 + p_4 + p_1 + p_3) \\ &= 4 \cdot 3 + 5 \cdot 7 + 1 \cdot 10 + 2 \cdot 12 \\ &= 81 \end{aligned}$$

Coincidentally, this solution is also an optimal solution to this example instance.

[1]Note that, while partial orders are defined to be reflexive, it does not make any sense to require a job to be completed before it can be started. Thus, we will often choose to omit reflexive edges in posets in the context of scheduling. This will be always clear from the context and will have no implications for our purposes.

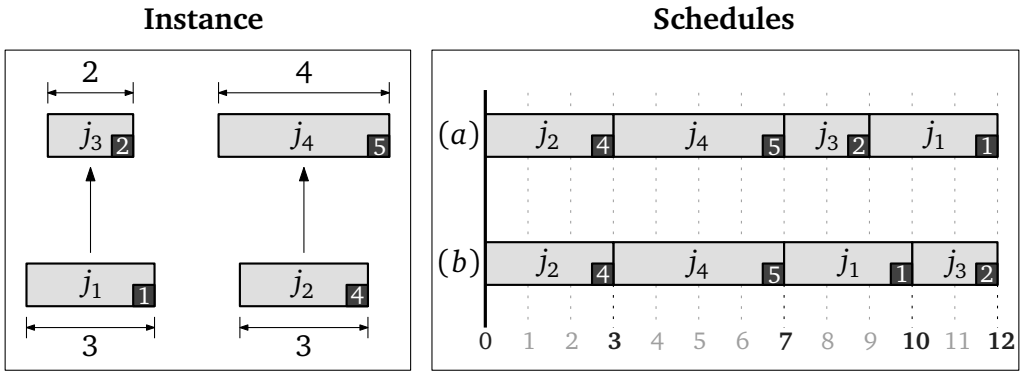


Figure 3.1. An example instance of problem $1|\text{prec}|\sum_j w_j C_j$ and two schedules: (a) an infeasible schedule and (b) a feasible (and optimal) schedule. Completion times of the feasible schedule (b) are in bold.

Inspecting the objective function a bit closer, we observe that there is a considerable portion of it that depends only on the definition of the instance. For example, a job's own processing time is always included in its completion time. Moreover, for any precedence constraint $(i, j) \in P$, the processing time p_i will be included in C_j for any feasible solution. We can rewrite the objective value as follows in order to separate this so-called *fixed cost* from the *variable cost*:

$$\begin{aligned}\sum_j w_j C_j &= \sum_j w_j \Big(p_j + \sum_{(i,j)\in L} p_i \Big) \\ &= \underbrace{\sum_j w_j p_j + \sum_{(i,j)\in P} w_j p_i}_{\text{fixed cost}} + \underbrace{\sum_{(i,j)\in L\setminus P} w_j p_i}_{\text{variable cost}}\end{aligned}$$

Clearly, since the fixed cost is present in any feasible solution, the problem can be reformulated as minimizing the variable cost. However, the fixed cost is important in the analysis of approximation algorithms, since it increases both the approximate solution value and the optimal value by an additive term, thus improving the approximation ratio. As an example, in the instance given above, the fixed cost is equal to

$$\sum_{i=1}^{4} p_i w_i + (p_1 \cdot w_3 + p_2 \cdot w_4) = 39 + 21 = 60$$

Thus the optimal solution L has variable cost 21. Assume an approximation algorithm that returns the solution $L_A = (j_1, j_2, j_3, j_4)$. The value of this solution

is

$$\text{val}(L_A) = 1 \cdot 3 + 4 \cdot 6 + 2 \cdot 8 + 5 \cdot 12 = 103 = 60 + \underbrace{43}_{\text{variable cost}}$$

The approximation factor of this algorithm is $103/81 \approx 5/4$ which is much better than the ratio $43/21 \approx 2$ given by comparing only the variable costs.

The impact of the fixed cost in the analysis of approximation algorithms is reflected throughout the history of this problem. For example, in [FM03] and [CM99] the authors show that any schedule that complies with the Sidney decomposition is a 2-approximation. However, the same is not true if only the variable cost is considered, as shown in [Uha08]. On the negative side, approximating the variable cost is known to be as hard as approximating the vertex cover problem [Sve08], whereas a similar result is not known to hold for the complete objective. We give an overview of the literature on this problem in the next Section.

3.1.1 Literature review

The computational complexity of the problem $1|\text{prec}|\sum_j w_j C_j$ has been of interest to researchers already in the seventies. It was shown to be strongly NP-hard in 1978 by Lawler [Law78] and Lenstra & Rinnooy Kan [LR78b]. This result implies that the problem does not allow for a fully polynomial time approximation scheme (FPTAS, see Theorem 1.3.4). Until recently, this was the best known inapproximability result for this problem (recent advances are described below).

On the positive side, several 2-approximation algorithms have been proposed [Pis92; Sch96b; HSSW97; CM99; CH99; FM03; Pis03]. We shall briefly outline the different approaches below. Schulz [Sch96b] gave 2-approximation algorithms using linear programming relaxations[2]. Chudak & Hochbaum [CH99] gave another algorithm based on a linear programming relaxation with two variables per constraint. Furthermore, they showed that their linear programming relaxation can be solved using one min-cut computation, making their algorithm combinatorial. Independently, Chekuri & Motwani [CM99] and Margot, Queyranne & Wang [FM03], provided identical, simple combinatorial 2-approximation algorithms based on Sidney's decomposition theorem [Sid75] from 1975.

A *Sidney decomposition* partitions the set N of jobs into sets $S_1, S_2, \ldots, S_k$ that can be scheduled in this order without loss of optimality. More precisely,

[2]This work later appeared in a joint journal version [HSSW97] together with the work of Hall, Shmoys, & Stein [HSW96], who used linear programming relaxations to give a constant factor $(4+\varepsilon)$-approximation algorithm for $1|\text{prec}|\sum_j w_j C_j$.

there exists an optimal schedule where jobs from S_i are processed before jobs from S_{i+1}, for any $i = 1, \ldots, k-1$. Lawler [Law78] showed that a Sidney decomposition can be computed in polynomial time by performing a sequence of min-cut computations. Chekuri & Motwani [CM99] and Margot, Queyranne & Wang [FM03] proved that every schedule that complies with a Sidney decomposition is a 2-approximate solution. As mentioned earlier, the fixed-cost is crucial for this analysis: a schedule that complies with a Sidney decomposition is not necessarily a 2-approximate solution if we only consider the variable-cost [Uha08]. Correa & Schulz [CS05] showed that all known 2-approximation algorithms follow a Sidney decomposition, and therefore belong to the class of approximation algorithms described by Chekuri & Motwani [CM99] and Margot, Queyranne & Wang [FM03]. Recently, Schulz & Uhan [SU08] showed for a large class of randomly generated instances that almost all instances are not Sidney decomposable. Hence, for almost all of those instances, any feasible schedule is a 2-approximation. They actually proved the stronger statement that for almost all *randomly* generated instances, all feasible schedules are arbitrarily close to optimal. Nevertheless, an algorithm that can *guarantee* a better than 2-approximate solution to any given instance remains unknown.

Due to the difficulty of obtaining better than 2-approximation algorithms for the general case, it is interesting to understand for which special cases one can achieve a better performance guarantee. A particularly successful and popular approach has been to consider special cases of precedence constraints. Indeed, as we sketched in section 2.1, Smith [Smi56] showed already in 1956 that, in the absence of precedence constraints[3], an optimal solution can be found by sequencing the jobs in non-increasing order of the ratio w_i/p_i. Later, several other results for special classes of precedence constraints were proposed (see [LLKS93] for a survey), most notably Lawler's [Law78] $O(n \log n)$ time algorithm for series-parallel precedence constraints. A nice alternative proof of the correctness of this algorithm was provided by Goemans & Williamson [GW00] who used the two-dimensional Gantt charts of Eastman et al. [EEI64]. For interval orders and convex bipartite precedence constraints, Woeginger [Woe03] gave approximation algorithms with an approximation ratio arbitrarily close to the golden ratio $\frac{1}{2}(1+\sqrt{5}) \approx 1.61803$. Using a similar approach, Kolliopoulos & Steiner [KS02] gave an approximation algorithm with the same performance guarantee (≈ 1.61803) for the special case of two-dimensional precedence constraints. This was later improved to a 3/2-approximation by Correa & Schulz [CS05].

[3] Or "whenever the precedence constraints form an antichain" (see Section 3.4)

Recently, Ambühl & Mastrolilli [AM09] settled an open problem first raised by Chudak & Hochbaum [CH99] and whose answer was subsequently conjectured by Correa & Schulz [CS05]. As shown by Correa & Schulz, the settlement of this conjecture has several interesting consequences for $1|\text{prec}|\sum_j w_j C_j$. The combined results of [CS05; AM09] imply the existence of an exact polynomial time algorithm for the special case of two-dimensional precedence constraints, a problem that previously was only known to be *approximable* in polynomial time [KS02; CS05], as mentioned above. Furthermore, it significantly generalized Lawler's exact algorithm for series-parallel orders [Law78].

The most significant implication of [CS05; AM09] is that $1|\text{prec}|\sum_j w_j C_j$ is in fact a special case of the weighted vertex cover problem. More precisely, they proved that every instance S of $1|\text{prec}|\sum_j w_j C_j$ can be translated in polynomial time into a weighted graph $G_{\mathbf{P}}^S$ (see Section 3.3 for details), such that finding an optimum of the variable-cost of S can be reduced to finding a minimum vertex cover in $G_{\mathbf{P}}^S$. This result even holds for approximate solutions: finding an α-approximate solution for the variable-cost of S can be reduced to finding an α-approximate vertex cover in $G_{\mathbf{P}}^S$. This result shed new light on the scheduling problem, as it provided simple explanations for this problem's behavior in terms of computational complexity. For instance, since the vertex cover problem can be approximated within a factor of 2, this seems to provide yet another simple 2-approximation algorithm for $1|\text{prec}|\sum_j w_j C_j$. However, it is remarkable that this approach is the first one to yield any constant approximation algorithm for the *variable* part, as results from [AM09; Uha08]. Moreover, the solvability of 2-dimensional precedence constraints can be gazed from the fact that the corresponding graph $G_{\mathbf{P}}^S$ becomes bipartite, as a classical result in graph theory states that vertex cover in bipartite graphs can be solved in polynomial time.

On the negative side, until recently the best known inapproximability result for this problem was its strong NP-completeness, i.e. the fact that it does not allow for an FPTAS (see Theorem 1.3.4). This still left open the possibility for arbitrarily good constant factor approximation algorithms. This was disproved recently by Ambühl et al. [AMS07] who showed that this problem does not have a PTAS, making a fairly standard assumption, namely that the satisfiability problem does not have subexponential algorithms (see [Sve08] for a detailed description).

The approximation gap of problem $1|\text{prec}|\sum_j w_j C_j$ remains one of the most perplexing questions in scheduling theory, and appears in the list of ten outstanding open problems in scheduling theory, compiled by Shuurman & Woeginger [SW99]. We know of several simple 2-approximation algorithms, but the

current techniques fail to provide any non-marginal inapproximability results. We are confident that a full understanding of the approximability of this problem will go hand in hand with a deeper understanding of approximability. A notable recent result by Bansal & Khot [Kho09] suggests that the above mentioned 2-approximation algorithms are best possible. Their result is based on a variant of the Unique Games Conjecture [Kho02], whose status remains open as of this writing. In contrast to the "P vs. NP"-conjecture, there seems to be no consensus in the community about the most probable outcome of this conjecture.

The contributions of this thesis to this problem are improvements whenever the precedence constraints contain some structure that can be exploited, and results in a characterization of the complexity of instances in terms of their precedence constraints.

3.2 A sketch of the algorithmic framework

In this Section we sketch the framework that we will present in the remainder of this chapter. Instead of giving an exact description of the individual components, this section conveys the intuition behind our approach, using the components as black boxes. We defer the details of the different steps to later sections, in which each component of our framework is examined individually. We revisit the framework in Section 3.5 after the components have been presented, and give some applications in Section 3.6. Figure 3.2 illustrates the different steps of our framework. The remaining of this section is to be read in parallel to Figure 3.2, to which we will make references in the text.

Assume that we are given an instance of problem $1|\text{prec}|\sum_j w_j C_j$. As discussed in Section 3.1, the objective value of any solution to this instance consists of an instance-dependent fixed cost and a solution-dependent variable cost. We can easily calculate the fixed cost of this instance and reformulate the problem as minimizing only the variable cost, as described above.

Now assume that we are given an algorithm (Step A in Figure 3.2) that creates an instance of the weighted vertex cover problem corresponding to the reformulated minimization problem above. In the following, we will try to approximate the vertex cover problem on the given graph.

First, assume that we are given a preprocessing procedure (Step B in Figure 3.2) that partitions the vertices of the graph into three sets, as follows:

- a set of vertices which we can choose in the vertex cover without loss of optimality

- a set of vertices which we can leave out of the vertex cover without loss of feasibility or optimality

- a set consisting of the remaining vertices, together with a guarantee that any optimal solution of the vertex cover in the subgraph induced by these nodes has value at least half of the their total weight W.

Due to the above procedure, we can concentrate on the subgraph induced by the third set of vertices. We will approximate the vertex cover problem on this graph using the following idea. Let us assume we are given an independent set in a graph. The complement of the independent set, i.e. all the vertices of the graph except the ones contained in the independent set, must form a vertex cover, for were there an uncovered edge in the graph, both of its endpoints would have to be in the independent set, contradicting its independence. This means that providing a big independent set automatically provides a small vertex cover, simply by complementation.

This redefines our goal as finding a "good" independent set, i.e. an independent set with a high total weight. We will do this using the following approach. Assume that we are given a coloring (Step C in Figure 3.2) of the graph, i.e. an assignment of colors to the vertices, such that adjacent vertices receive different colors[4]. Looking at a color class, i.e. a set of vertices all of which were assigned the same color, we observe that they must form an independent set, by definition of the coloring. Thus, a coloring with t colors provides t disjoint independent sets that cover the vertices of the graph.

It is natural to choose the color class with the highest total weight to be the independent set used in defining the vertex cover solution by complementation. Since the color classes cover all the vertices, the weight of the heaviest independent set can be lower bounded by the average weight of the color classes, which is W/t. The vertex cover defined by complementation (Step D in Figure 3.2) will thus have weight the remaining $W - W/t = (1 - 1/t)W$.

Recall that, due to the preprocessing procedure applied to the original graph above we can assume that the optimal vertex cover for this subgraph has total weight at least $W/2$. Therefore, providing one of weight $(1 - 1/t)W$ constitutes a $2 - 2/t$-approximation.

We have now approximated the vertex cover problem in the subgraph with a ratio of $2-2/t$. In order to achieve this approximation guarantee for the scheduling problem, we need to roll back the previous steps in reverse order. First, let

[4]Note that in Section 3.5.2, when we revisit the framework sketched here, we will extend this approach to *fractional* coloring. We only mention the integral coloring here for simplicity.

us add the first set of vertices given by the preprocessing that we know we can add without loss of optimality. It is intuitive that, after performing this step, the impact of the part of the graph in which our solution is suboptimal decreases. Thus, the approximation guarantee still holds (and actually might improve after this step). This means that we can guarantee a $(2-2/t)$-approximation for the vertex cover of the *whole* graph.

Now, assume that we were given an algorithm (Step E in Figure 3.2) that transforms an approximate solution to the vertex cover of this graph into an approximate solution to the variable part of the scheduling problem, such that the approximation factor remains the same or improves. Applying this algorithm on the $(2-2/t)$-approximate solution constructed above gives a solution to the scheduling problem with a $(2-2/t)$-approximation guarantee for the variable part.

The last step is obvious: we need to analyze the solution constructed in terms of the *whole* objective function, not just the variable part. As mentioned in Section 3.1, the analysis of the approximation ratio can only profit when the fixed cost of the instance is taken into consideration. Overall, this provides a $(2-2/t)$-approximation for the scheduling problem.

We briefly mention the content of the different components that were used as black boxes above, before describing them individually in detail in the following sections.

Step A is a result of a series of linear programming formulations of the problem $1|\text{prec}|\sum_j w_j C_j$. Based on a formulation by Potts [Pot80] and relaxations of Chudak & Hochbaum [CH99] and Correa & Schulz [CS05], the variable part of any instance of $1|\text{prec}|\sum_j w_j C_j$ can be associated with an instance of the weighted vertex cover graph (see Section 3.3)

Step B is a classical result by Nemhauser & Trotter [NT75], showing the half-integrality of a linear programming formulation for the vertex cover problem. We describe their contributions in Section 3.3

Step C is the core of our contribution. We use the dimension theory of partial orders in order to color the graph with few colors, whenever the partial order has low dimension. We point out that this component can be significantly improved by considering the fractional dimension and fractional coloring, yielding a new framework with a wider range of applications. This will be described in Section 3.5 and applied in Section 3.6. We sketch the simpler framework based on (integral) dimension and coloring in this Section for the sake of simplicity.

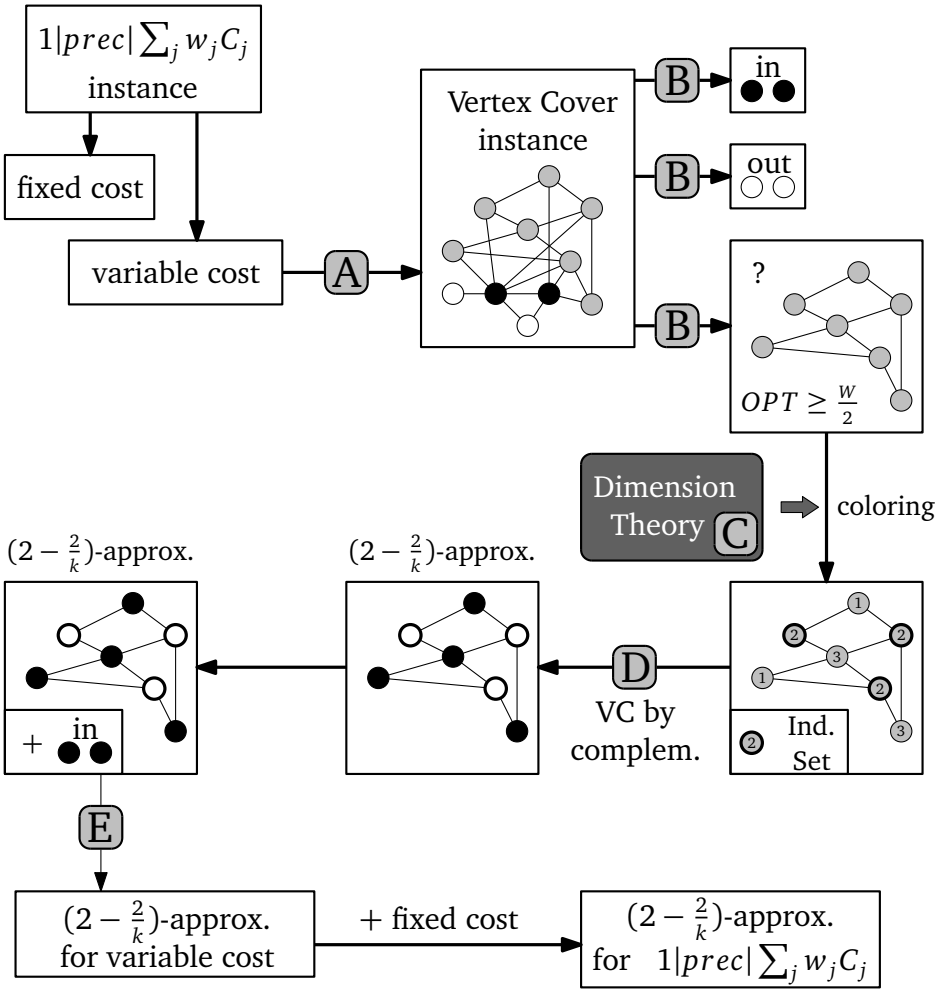


Figure 3.2. Sketch of the framework.

Step D is due to Hochbaum [Hoc83], who gave a $(2-2/t)$ approximation algorithm for the vertex cover problem, whenever the graph can be colored with t colors, and is discussed in Section 3.3

Step E is due to a combination of results by Correa & Schulz [CS05] and Ambühl & Mastrolilli [AM09] which yield a polynomial time algorithm that converts an approximate solution to the vertex cover problem into an approximate solution to the variable part of 1|prec|$\sum_j w_j C_j$, without deteriorating the objective value. This is described in Section 3.3.

3.3 Single Machine Scheduling and Vertex Cover

We first give an overview of the vertex cover problem, before discussing its relationship with the scheduling problem.

3.3.1 The Vertex Cover problem

The VERTEX COVER problem (see e.g. [Hoc95]) is arguably one of the most prominent NP-complete problems in Graph Theory. It is among the 21 NP-complete combinatorial problems in the landmark paper of Richard Karp [Kar72]. Moreover, it is a problem that illustrates the limits of the current theory on Approximation Algorithms, since it is a very basic problem whose approximation gap remains open as of this writing (see below). Its notoriety can be glanced at by the fact that it is by far the most accessed entry in the on-line compendium of Crescenzi & Kann [CK98]. Besides its prominence in Complexity Theory, the VERTEX COVER problem is also important in practice, as it can naturally arise when a set of dependent entities is to be covered using a minimum amount of resources. A formal definition follows:

Problem 3.3.1 (VERTEX COVER)

Given: *A Graph $G(V,E)$ with vertex set V and edge set E, and a weight function $w : V \to \mathbb{Q}, i \mapsto w_i$.*

Find: *A* vertex cover *that minimizes the total weight, i.e. a subset $S \subseteq V$ such that for each edge $e = \{v,w\} \in E$, we have $|S \cap \{v,w\}| \geq 1$ and $\sum_{v \in S} w_v$ is minimized.*

There are several 2-approximation algorithms for this problem (see [Pas97] for a survey). We describe one of them, due to Nemhauser & Trotter, in the next section. On the negative side, Dinur & Safra [DS02] showed that the minimum vertex cover cannot be approximated within a factor of 1.3606 for any sufficiently large vertex degree unless P=NP. This leaves the interval between 1.3606 and 2 as the approximability gap for this problem. We note that this gap vanishes, if one is willing to assume the Unique Games Conjecture [KR08].

Half-integrality and a simple 2-approximation

Consider the following Integer Programming (IP) formulation for the vertex cover problem on a given graph $G(V,E)$.

$$\begin{aligned} \text{[VC-IP]} \qquad \min \quad & \sum_{i \in V} w_i x_i \\ \text{s.t.} \quad & x_i + x_j \geq 1, \quad \{i,j\} \in E \\ & x_i \in \{0,1\}, \quad i \in V \end{aligned}$$

That is, to each vertex $i \in V$ of the graph we associate an indication variable x_i that can take two values, 0 and 1. The interpretation of these values will be that $x_i = 1$ corresponds to vertex i being *picked* in the vertex cover solution, while $x_i = 0$ corresponds to i *not being picked*. With this interpretations, the constraints that at least one endpoint is to be picked in the vertex cover for each edge $e = \{i,j\}$ can be expressed as $x_i + x_j \geq 1$. Finally, the value of $\sum_{i \in V} w_i x_i$ gives the total sum of picked vertices.

Since the [VC-IP] is an exact formulation of the vertex cover problem, it is NP-hard to solve. Towards a tractable (but not exact) formulation, we relax the integrality constraints $x_i \in \{0,1\}$ by replacing them by $0 \leq x_i \leq 1$ (note that the upper bound is redundant since it is implied by the combination of the constraints with the objective function). This gives the following Linear Programming (LP) relaxation.

$$\begin{aligned} \text{[VC-LP]} \qquad \min \quad & \sum_{i \in V} w_i x_i \\ \text{s.t.} \quad & x_i + x_j \geq 1, \quad \{i,j\} \in E \\ & x_i \geq 0, \quad i \in V \end{aligned}$$

In 1973, Nemhauser & Trotter [NT73; NT75] studied the above two programs and proved that the linear program can be solved combinatorially in polynomial time, i.e. without invoking a call to an LP-solver. They also proved that any basic feasible solution for [VC-LP] is *half-integral*, that is $x_i \in \{0, 1/2, 1\}$ for all $i \in V$. This partitions the vertices into three sets $V_0, V_{1/2}$ and V_1 according to their value. In the following we will denote by W_i the total weight of vertices in V_i for $i \in \{0, 1/2, 1\}$.

Observe that there is no edge inside V_0 or between V_0 and $V_{1/2}$, by feasibility of the half-integral solution. In other words, the set V_1 separates V_0 (which is an independent set) from $V_{1/2}$. This is particularly useful, due to the *persistency property* [NT73; NT75]: there exists an optimal solution S^*_{per} to the (integral) problem such that all vertices in V_1 are picked and all the vertices in V_0 are not

picked. This allows us to concentrate on the graph $G[V_{1/2}]$, i.e. the subgraph of G induced by the vertices in $V_{1/2}$: any vertex cover of $G[V_{1/2}]$ can be completed into a vertex cover of G simply by adding the vertices in V_1. Moreover, since by the addition of W_1 into the objective function, the impact of $W_{1/2}$ decreases, any α-approximation guarantee for a vertex cover in $G[V_{1/2}]$ carries over to the graph G. This preprocessing is illustrated in Figure 3.3.

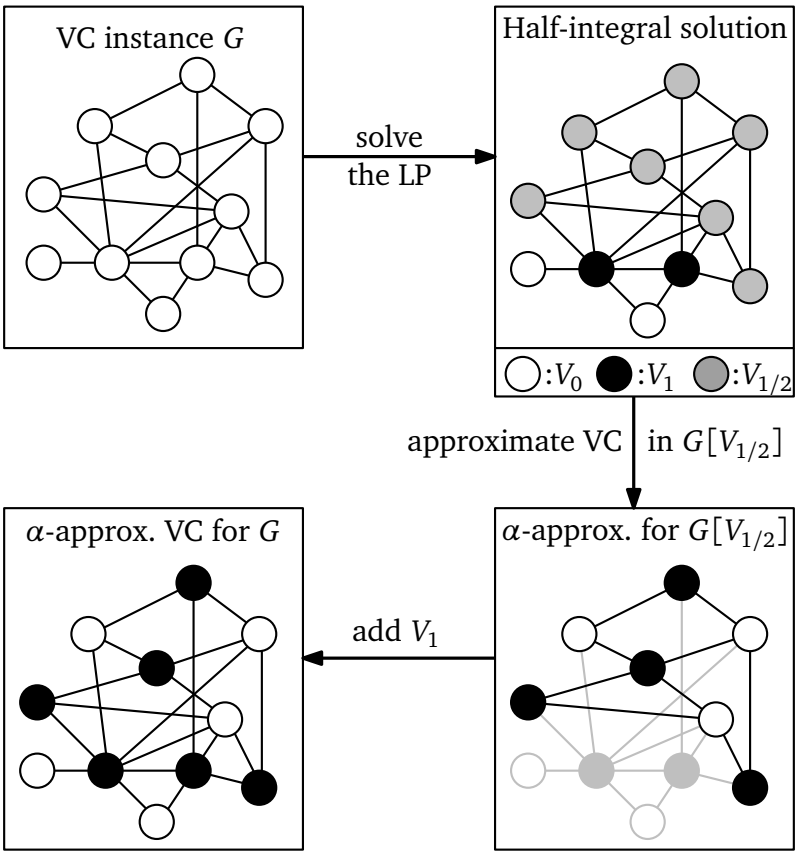


Figure 3.3. The Nemhauser & Trotter preprocessing [NT75; NT73] allows us to concentrate on the graph $G[V_{1/2}]$, when looking for a vertex cover of G.

Note that, since [VC-LP] is a relaxation of the vertex cover problem, the value OPT_{LP} of an optimal half-integral solution S^*_{LP} of [VC-LP] is "too optimistic", i.e. it provides a lower bound on the value of any feasible solution. Let us compare the value OPT_{LP} to the value OPT^*_{per} of a persistent solution S^*_{per} mentioned above. The two solutions perform equally well on the sets of vertices V_0 and V_1, since they are identical on these sets of vertices. Thus, for OPT_{LP} to

be a lower bound of OPT^*_{per}, it needs to be a lower bound when the two solutions are restricted to the set $V_{1/2}$. Thus, any vertex cover of $G[V_{1/2}]$, and in particular an optimal one, has value at least

$$\text{val}(S^*_{LP}[V_{1/2}]) = \sum_{i \in V_{1/2}} x_i w_i = \frac{1}{2} \sum_{i \in V_{1/2}} w_i = \frac{W_{1/2}}{2}$$

An obvious way of solving the vertex cover problem in $G[V_{1/2}]$ would be to pick all the nodes. Surprisingly, this trivial algorithm already provides a 2-approximation for this problem. Indeed, as discussed above, a persistent optimal solution will have value at least $W_{1/2}/2 + W_1$ while the trivial solution has value $W_{1/2} + W_1$, giving an approximation ratio of at most 2. The next section suggests a better way of dealing with the subgraph $G[V_{1/2}]$.

A better approximation based on coloring

Hochbaum [Hoc83] suggested a better way of computing the vertex cover in the subgraph $G[V_{1/2}]$, based on the graph coloring problem, defined below. Her approach yields better than 2-approximation algorithms whenever a coloring with a small number of colors is provided for the input graph.

Problem 3.3.2 (GRAPH COLORING)

Given: *A graph $G(V, E)$ with vertex set V and edge set E.*

Find: *A k-*coloring*, i.e. an assignment of colors to the vertices of the graph from a palette of size k such that no two adjacent vertices share the same color and k is minimized.*

The smallest k for which it is possible to find a k-coloring is called the *chromatic number* of the graph G, denoted by $\chi(G)$.Let us define the set of vertices that were assigned color i in a feasible coloring of G to be the *color class* i. Observe that any color class must be an independent set in G. This is because if there were an edge between two vertices that belong to the same color class, it would imply that those two vertices were assigned the same color despite being adjacent, contradicting the feasibility of the coloring.

Assuming that a k-coloring of the graph is given, a vertex cover can be constructed by ordering the color classes by their total weight (i.e. the sum of weights of the vertices in the color class) and picking all color classes except the heaviest one to be in the vertex cover solution. Since the omitted vertices

form an independent set, this solution is feasible. Furthermore, since the heaviest color class has total weight at least the average among the color classes, i.e. $W_{1/2}/k$, the total weight of the vertex cover in $G[V_{1/2}]$ is at most

$$W_{1/2} - \frac{W_{1/2}}{k} = \left(1 - \frac{1}{k}\right) W_{1/2}.$$

As hinted to in Section 3.2, this approach together with the fact that any feasible vertex cover for $G[V_{1/2}]$ has weight at least $W_{1/2}/2$ leads to an approximation ratio at most

$$\frac{W_{1/2}(1-1/k)}{W_{1/2}/2} = 2 - \frac{2}{k}.$$

We demonstrate this approach in Figure 3.4 giving an example of a 3-colorable graph.

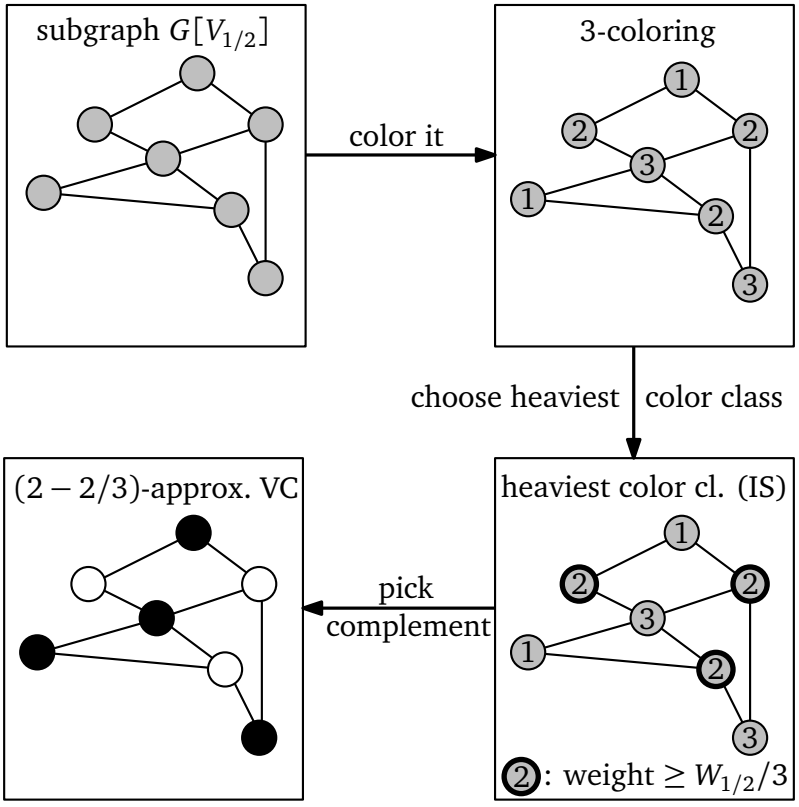


Figure 3.4. Hochbaum's approach to approximate the vertex cover problem in $G[V_{1/2}]$ using graph coloring.

Fractional coloring and Vertex Cover

In this Section we take the approach of [Hoc83] a step further and extend it to the case when a *fractional* coloring [SU97] of the graph is known. Let us first give a definition.

Problem 3.3.3 (Fractional Graph Coloring)

Given: *A graph $G(V,E)$ with vertex set V and edge set E.*

Find: *An a:b-coloring, i.e. an assignment of a* set *of b colors to each vertex of the graph from a palette of size a such that no two adjacent vertices share a color and a/b is minimized.*

We call a coloring that assigns b colors to each vertex a *b-fold coloring* with a 1-fold coloring being the traditional coloring described above, see Problem 3.3.2. Similarly, the smallest integer a for which a b-fold coloring exists is called the b-fold chromatic number, denoted $\chi_b(G)$. Now, the *fractional chromatic number* $\chi_f(G)$ of a graph is given by the limit[5]

$$\chi_f(G) = \lim_{t\to\infty} \frac{\chi_b(G)}{b} = \inf_b \frac{\chi_b(G)}{b}.$$

Observe that the chromatic number is always an upper bound on the fractional chromatic number, since a (traditional) k-coloring can be turned into a $(c\cdot k)$:c-coloring simply by replicating colors.

The color classes in a fractional coloring are still independent sets, only that now they do not form a partitioning of the nodes, but overlap. More precisely, every vertex of the graph in a b-fold coloring appears in the b color classes corresponding to the colors assigned to it. Again, we can construct a vertex cover solution by complementing the heaviest among the a color classes.

The power of this new approach is best illustrated with an example. Consider the graph C_5, i.e. a circle on 5 nodes, with arbitrary weights. Since this graph is an odd cycle, its chromatic number is $\chi(C_5) = 3$ (see Figure 3.5a). On the other hand, we can find a 5:2-coloring for C_5, as shown in Figure 3.5b.

Let W be the total weight of the graph and $W[i]$ the weight of color class i. As shown above, if it can be assumed that any vertex cover of the graph has weight at least $W/2$, Hochbaum's approach [Hoc83] yields an approximation guarantee of $2 - 2/3 = 4/3$, given a 3-coloring of the graph. If instead we

[5] That this limit exists follows from the subadditivity of the t-fold chromatic number, i.e. $\chi_{b+c}(G) \le \chi_b(G) + \chi_c(G)$

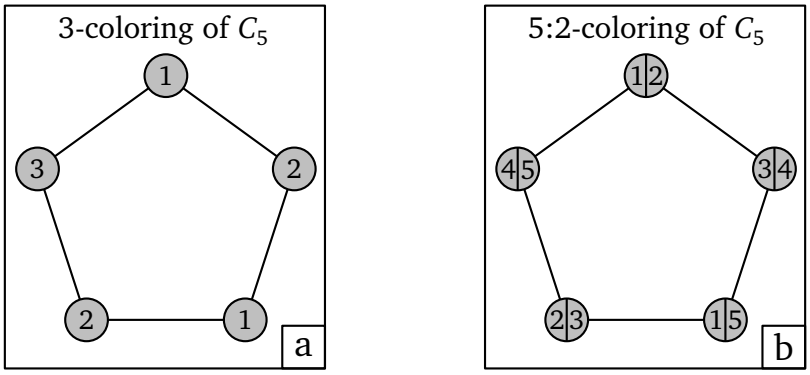


Figure 3.5. On the left a 3-coloring of C_5. On the right a 2-fold coloring using a palette of size only 5 (as opposed to the straight-forward palette of 6 colors resulting by replication).

choose the heaviest color class in the 5 : 2-coloring, using again an averaging argument we get that

$$\max_{1\leq i\leq 5}\{W[i]\} \geq \frac{\sum_{i=1}^{5} W[i]}{5} = \frac{2W}{5}$$

where the last equality holds because each vertex appears in two color classes. Thus, using fractional coloring we get an approximation guarantee of at most

$$\frac{W - 2W/5}{W/2} = \frac{6}{5},$$

an improvement over the 4/3-approximation given by [Hoc83].

3.3.2 Connection between $1|\text{prec}|\sum_j w_j C_j$ and Vertex Cover

In a series of recent papers [CH99; CS05; AM09] it was proved that the problem $1|\text{prec}|\sum_j w_j C_j$ is a special case of the minimum weighted vertex cover problem. In this section we give an overview of how this result was obtained. In Section 3.5.1 we make the connection between the scheduling problem and dimension theory explicit by pointing out that the vertex cover graph obtained from a scheduling instance, with precedence constraints in the form of a poset $\mathbf{P}$, is in fact the graph of incomparable pairs $G_{\mathbf{P}}$, defined in dimension theory of partial orders (see Section 3.4).

To simplify notation, we implicitly assume hereafter that tuples and sets of jobs have no multiplicity. Therefore, $(a_1, a_2, \ldots, a_k) \in N^k$ and $\{b_1, b_2, \ldots, b_k\} \subseteq N$ denote a tuple and a set, respectively, with k distinct elements.

In the following, we introduce several linear programming formulations and relaxations of $1|\text{prec}|\sum_j w_j C_j$ using linear ordering variables δ_{ij}. The variable δ_{ij} has value 1 if job i precedes job j in the corresponding schedule, and 0 otherwise. In this sense, the variables δ_{ij} and δ_{ji} are complementary to each other.

The following linear programming formulation of $1|\text{prec}|\sum_j w_j C_j$ using linear ordering variables is due to Potts [Pot80]:

$$\text{[P-IP]} \qquad \min \quad \sum_{j \in N} p_j w_j + \sum_{(i,j) \in N^2} \delta_{ij} p_i w_j \tag{3.1a}$$

$$\text{s.t.} \quad \delta_{ij} + \delta_{ji} = 1 \qquad \{i,j\} \subseteq N \tag{3.1b}$$

$$\delta_{ij} = 1 \qquad (i,j) \in P \tag{3.1c}$$

$$\delta_{ij} + \delta_{jk} + \delta_{ki} \leq 2 \qquad (i,j,k) \in N^3 \tag{3.1d}$$

$$\delta_{ij} \in \{0,1\} \qquad (i,j) \in N^2 \tag{3.1e}$$

Constraint (3.1b) ensures that δ_{ij} and δ_{ji} are complementary, meaning that for any two jobs $\{i,j\}$ either job i is scheduled before j or vice versa. If job i is constrained to precede j in the partial order P, then this is seized by constraint (4.3d). The set of constraints (3.1d) is used to capture the transitivity of the ordering relations (i.e., if i is scheduled before j and j before k, then i is scheduled before k). It is easy to see that [P-IP] is indeed an exact formulation of the problem $1|\text{prec}|\sum_j w_j C_j$.

Chudak & Hochbaum [CH99] suggested to study the following relaxation of [P-IP] in which the transitivity constraints are modified:

$$\text{[CH-IP]} \qquad \min \quad \sum_{j \in N} p_j w_j + \sum_{(i,j) \in N^2} \delta_{ij} p_i w_j \tag{3.2a}$$

$$\text{s.t.} \quad \delta_{ij} + \delta_{ji} = 1 \qquad \{i,j\} \subseteq N \tag{3.2b}$$

$$\delta_{ij} = 1 \qquad (i,j) \in P \tag{3.2c}$$

$$\delta_{jk} + \delta_{ki} \leq 1 \qquad (i,j) \in P, \{i,j,k\} \subseteq N \tag{3.2d}$$

$$\delta_{ij} \in \{0,1\} \qquad (i,j) \in N^2 \tag{3.2e}$$

In [CH-IP], the set of constraints (3.1d) from [P-IP] are replaced by the set of constraints (3.2d). These inequalities correspond in general to a proper subset

of (3.2d), since the transitivity constraints (3.2d) are dropped, unless there is a precedence constraint between two of the participating jobs.

Correa & Schulz [CS05] proposed the following integer programming formulation [CS-IP], which is a relaxation of [P-IP]. For $(i,j) \in N^2$, the term[6] $i \| j$ means that $(i,j) \notin P$ and $(j,i) \notin P$. Note that in [CS-IP], we only need variables δ_{ij} for jobs $i \| j$.

$$\text{[CS-IP]} \qquad \min \quad \sum_{j \in N} p_j w_j + \sum_{(i,j) \in P} p_i w_j + \sum_{i \| j} \delta_{ij} p_i w_j \tag{3.3a}$$

$$\text{s.t.} \quad \delta_{ij} + \delta_{ji} \geq 1 \qquad i \| j \tag{3.3b}$$

$$\delta_{ik} + \delta_{kj} \geq 1 \qquad (i,j) \in P, j \| k, k \| i \tag{3.3c}$$

$$\delta_{i\ell} + \delta_{kj} \geq 1 \qquad (i,j) \in P, j \| k, (k,\ell) \in P, \ell \| i \tag{3.3d}$$

$$\delta_{ij} \in \{0,1\} \qquad (i,j) \in N^2, i \| j \tag{3.3e}$$

Note that in this formulation, the objective function has been split into the fixed-cost $\sum_{(i,j) \in P} p_i w_j + \sum_{i \in N} p_i w_i$ and the variable-cost $\sum_{(i,j) \in L \setminus P} p_i w_j$ defined in Section 3.1, where $L = \{(i,j) : \delta_{ij} = 1\}$ is not necessarily a total ordering of the jobs. As discussed above, the fixed part can be ignored during optimization.

There is a strong resemblance of [CS-IP] to the [VC-IP] given in Section 3.3.1, once the fixed part of (3.3a) is put aside. Indeed, [CS-IP] can be interpreted as a vertex cover problem of the following graph:

Vertices There is a vertex for every (ordered) pair of jobs $(i,j), i \| j$, with δ_{ij} being the associated indication variable.

Weights The weight of the vertex corresponding to the pair (i,j) is given by the coefficient of δ_{ij} in the variable part of the objective function (3.3a) and is equal to the product $p_i w_j$.

Edges There is an edge between two ordered pairs if their corresponding ordering variables appear together in one of the constraints (3.3b), (3.3c) and (3.3d).

Correa & Schulz [CS05] also proved that their formulation [CS-IP] is equivalent to [CH-IP] and they conjectured that an optimal solution to $1|\text{prec}|\sum_j w_j C_j$ gives an optimal solution to [CH-IP] as well. The conjecture in [CS05] was recently settled in the affirmative by Ambühl & Mastrolilli [AM09], who proved that any feasible solution to [CH-IP] can be turned into a feasible solution

[6] Such two jobs are called an "incomparable pair" in poset terminology

to $1|\text{prec}|\sum_j w_j C_j$ without deteriorating the objective value and in polynomial time. As the fixed-cost remains unaffected by these transformations, the results in [CS05; AM09] imply that the problem of minimizing the variable-cost of $1|prec|\sum_j w_j C_j$ is a special case of minimum weighted vertex cover.

For a scheduling instance S with precedence constraints $\mathbf{P} = (N, P)$, we let $G^S_{\mathbf{P}}$ be the vertex cover graph obtained by interpreting [CS-IP] as a vertex cover problem, as described above. The following theorem summarizes the aforementioned results.

Theorem 3.3.4 ([CH99; CS05; AM09]) *Let S be an instance of $1|prec|\sum_j w_j C_j$ with precedence constraints $\mathbf{P}$. Then an α-approximate solution to the weighted vertex cover problem on $G^S_{\mathbf{P}}$ can, in polynomial time, be turned into an α-approximate solution to the variable-cost part of S.*

As the approximation guarantee can only improve by taking into account the fixed-cost, we have that an α-approximation algorithm for the weighted vertex cover problem associated to $1|prec|\sum_j w_j C_j$, yields *at least* an α-approximation algorithm for the scheduling problem.

However, as mentioned earlier, it is unknown whether the vertex cover problem can be approximated with a ratio better than 2 in general graphs, and there is some evidence against the existence of such an algorithm [KR08]. Thus, the above connection by itself only provides another 2-approximation algorithm for this problem. The decisive point is that the graph $G^S_{\mathbf{P}}$ obtained from $1|\text{prec}|\sum_j w_j C_j$ exhibits some nice structure, leading to better approximation algorithms for several families of precedence constraints. In fact, it turns out that it coincides with a graph well known to Combinatorialists: the graph of incomparable pairs $G_{\mathbf{P}}$ of partial orders (see Section 3.4.1).

3.4 Dimension Theory of Partial Orders

A *partially ordered set* (or *poset*) formalizes the intuitive concept of an ordering, sequencing, or arrangement of the elements of a set. The theory of partially ordered sets is central to combinatorics and arises in many different contexts. We are going to introduce it by giving an example.

Consider the set $A = \{a, b, c, d\}$ and a set N consisting of five subsets of A, namely $S_1 = \{a\}$, $S_2 = \{b\}$, $S_3 = \{c\}$, $S_4 = \{a, b, c\}$ and $S_5 = \{b, d\}$. The elements of the set N, called the *groundset*, can be partially ordered by the reflexive, antisymmetric and transitive relation P which we define, in the example, by use of the set inclusion "$\subseteq$" (see also Figure 3.6): for any $S, S' \in N$ we have

$(S, S') \in P$ if and only if $S \subseteq S'$. To emphasize the order concept, we write $x \leq y$ when $(x, y) \in P$.

For any $S, S' \in N$, in case $S' \leq S$ or $S \leq S'$, we will say that the (ordered) pair (S, S') is *comparable*, otherwise it is called *incomparable* (for $S \neq S'$ we call S' *larger than* S, if $S \leq S'$, also denoted by $S < S'$). For instance, S_1 and S_4 are comparable since $S_1 \leq S_4$, whereas neither $S_1 \leq S_5$ nor $S_5 \leq S_1$ holds which makes the pairs (S_1, S_5) and (S_5, S_1) incomparable. The groundset N together with the partial order P define a *partially ordered set (poset)* $\mathbf{P} = (N, P)$. We denote the set of all incomparable points of a poset $\mathbf{P}$ by $inc(\mathbf{P})$. A poset that does not have any incomparable pairs is called a *linear order*.

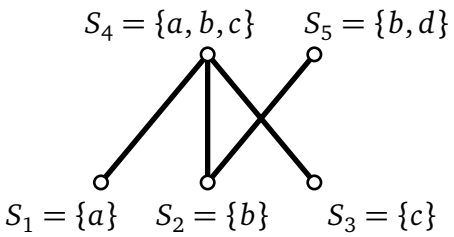


Figure 3.6. The Hasse diagram of the example poset $\mathbf{P}$ (each element of N is represented by a vertex on the plane and an upwards going line segment is drawn from x to y whenever $x < y$, unless there is a $z \in N$ such that $x < z < y$).

An *extension* of a poset $\mathbf{P} = (N, P)$ is any poset $\mathbf{P}' = (N, P')$ such that $P \subseteq P'$. Moreover, if P' is a linear order, we call $\mathbf{P}'$ a *linear extension* of $\mathbf{P}$. Putting aside the set-inclusion interpretation of the example poset, we can construct an extension of $\mathbf{P}$ by adding, for example, $S_3 \leq S_5$ to P. The resulting poset $\mathbf{P}'$ is depicted in Figure 3.7. We say that the extension $\mathbf{P}'$ *reverses* the (ordered) incomparable pair (S_5, S_3).

Constructing a linear extension that satisfies certain properties is central to several problems in combinatorics. The scheduling problem $1|prec|\sum_j w_j C_j$ falls in this category. The construction of linear extensions of a poset also arises when determining its dimension, as explained in the following.

For a family $\mathscr{R}$ of linear extensions of P, we call $\mathscr{R}$ a *realizer* of P if $\bigcap \mathscr{R} = \mathbf{P}$, i.e. for all $a, b, \in N$, $a \leq b$ in P if and only if $a \leq b$ in every $L \in \mathscr{R}$. Note that for each incomparable pair $(a, b) \in \text{inc}(\mathbf{P})$, there must be at least one linear extension in the realizer that reverses it. Thus, for any two $a \| b$, there must be a linear extension in the realizer containing (a, b) and another one containing (b, a). This means that, unless P is already a linear ordering or $P = \emptyset$, any real-

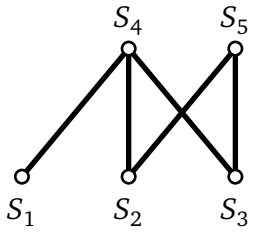
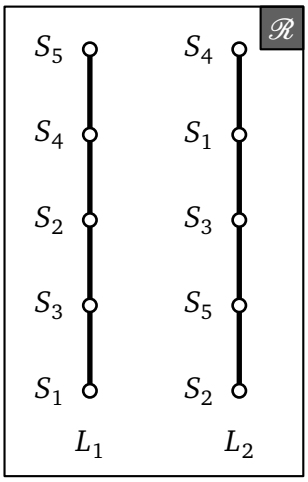


Figure 3.7. The Hasse diagram of the poset $\mathbf{P}'(N, P')$, an extension of the poset $\mathbf{P}(N, P)$ with $P' = P \cup \{(S_3, S_5)\}$

izer will have size at least 2. The least positive integer t for which there is such a realizer $\mathscr{R} = \{L_1, \ldots, L_t\}$ of $\mathbf{P}$ is called the *dimension* of $\mathbf{P}$, denoted by $\dim(\mathbf{P})$. A realizer of size t will be abbreviated by the name *t-realizer*. Figure 3.8 shows that the dimension of the example poset is at most two (and thus equal to two) by giving a 2-realizer, together with a table containing, for each incomparable pair (a, b) a linear extension in which it is reversed.



a \ b	S_1	S_2	S_3	S_4	S_5
S_1		L_2	L_2		L_2
S_2	L_1		L_1		
S_3	L_1	L_2			L_2
S_4					L_2
S_5	L_1		L_1	L_1	

Figure 3.8. A 2-realizer for the example poset of Figure 3.6. In the table on the right, a cell corresponding to an incomparable pair (a, b) contains a linear extension of the realizer in which it is reversed.

Similar to the coloring problem, also the dimension has a fractional counterpart. A *k:t-realizer* is defined as a multiset of t linear extensions $\mathscr{R} = \{L_1, \ldots, L_t\}$

in which each incomparable pair is reversed at least k times[7]. The *fractional dimension* of $\mathbf{P}$, denoted by fdim($\mathbf{P}$), is the infimum of the set of ratios t/k for which there exist k:t-realizers [BS92]. Since a t-realizer can be seen as a $1:t$-realizer, the dimension is a natural upper bound on the fractional dimension.

A natural question is for which posets one can construct a t-realizer in polynomial time. In the general case, Yannakakis [Yan82] proved that determining whether the dimension of a poset is at most t is NP-complete for every $t \geq 3$. Moreover, Hegde & Jain [HJ07] recently proved that it is hard to approximate the (fractional) dimension of a poset with n elements within a factor $n^{0.5-\varepsilon}$, in the general case. However, for several special cases, a minimal realizer can be computed in polynomial time (see e.g. [Tro92; Möh89]).

3.4.1 The Hypergraph of Incomparable Pairs

It is easy to see that, when constructing an extension of $\mathbf{P}$, there are some groups of incomparable pairs that cannot be reversed at the same time. Obviously, an extension of $\mathbf{P}$ cannot reverse both (S_3, S_5) and (S_5, S_3) at the same time. This implies that, unless a poset is already a linear order, any realizer needs to contain more than one linear extensions in order to reverse every incomparable pair at least once. For the pairs of incomparable pairs mentioned above, it is obvious that they cannot be reversed at the same time. However, there are also less obvious pairs of incomparable pairs for which this is true. By examining the Hasse-diagram of $\mathbf{P}$ (see Figure 3.6), one can conclude that reversing both (S_2, S_1) and (S_1, S_5) would lead to an inconsistency, i.e. a "cycle" in the ordering: adding $S_1 \leq S_2$ and using transitivity leads to $S_1 \leq S_2 \leq S_5$, which contradicts $S_5 \leq S_1$. In general, there can also be groups bigger than two pairs that cannot be all reversed at the same time without introducing contradictions (e.g., (S_2, S_1), (S_1, S_3) and (S_3, S_5)).

The above observations naturally lead to the definition of the *hypergraph of incomparable pairs* $\mathbf{H_P}$ of a poset $\mathbf{P}$ [FT00] defined as follows. The vertices of $\mathbf{H_P}$ are the incomparable pairs in $\mathbf{P}$. The edge set consists of those sets U of incomparable pairs such that:

1. No linear extension of $\mathbf{P}$ reverses all incomparable pairs in U.

[7] Note the unfortunate dissonance in the notation of fractional dimension and fractional coloring literature: the total number of colors a is denoted *first* in an a:b-coloring, while the total number of linear extensions t is is denoted *second* in a k:t-realizer. To avoid confusion, the reader is advised to keep in mind that these fraction cannot be less than one.

2. For every proper subset of U there is a linear extension that reverses all incomparable pairs in U.

Figure 3.9 depicts the hypergraph of incomparable pairs for our example poset. The graph resulting from $\mathbf{H_P}$ by restricting the set E to hyperedges of size 2 (i.e. graph edges), is called the *graph of incomparable pairs* and denoted by $G_\mathbf{P}$.

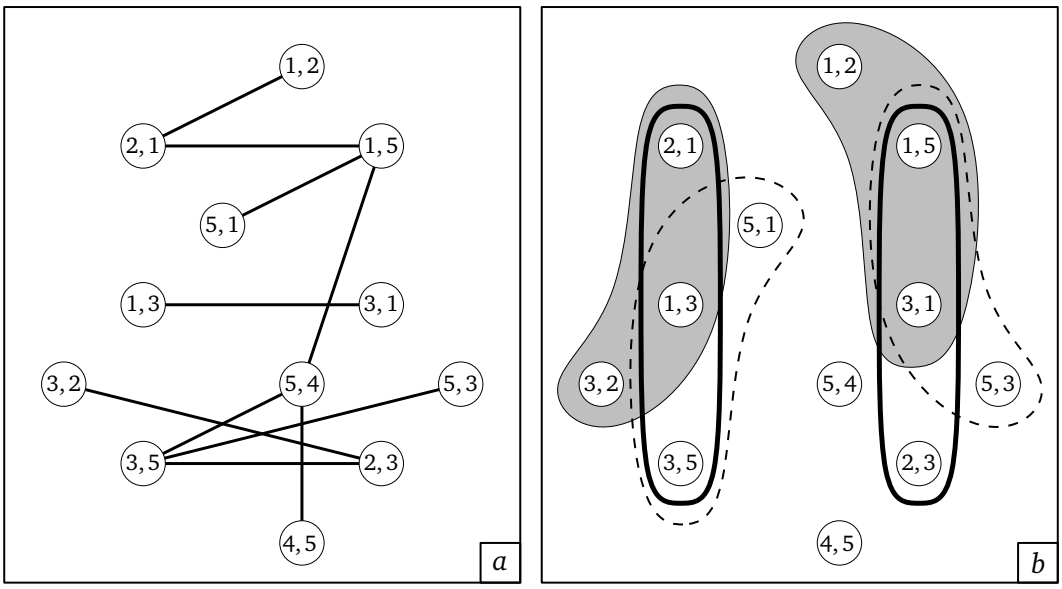


Figure 3.9. The graph $G_\mathbf{P}$ and the hypergraph $\mathbf{H_P}$ of incomparable pairs of our example poset $\mathbf{P}$. The *graph* of incomparable pairs $G_\mathbf{P}$ is depicted on the left. The *hypergraph* $\mathbf{H_P}$ can be obtained by adding the hyperedges on the right to the edges on the left.

It turns out that properties of this graph reflect properties of the originating poset. More precisely, the dimension of $\mathbf{P}$ is equal to the chromatic number of the hypergraph $\mathbf{H_P}$ [Tro92]. Brightwell & Scheinerman [BS92] showed that the fractional counterparts of these values also coincide: the fractional dimension of $\mathbf{P}$ equals the fractional chromatic number of the hypergraph $\mathbf{H_P}$. We include the proof of these statements below, as they offer useful insights into the (hyper)graph of incomparable pairs. Since dimension is a special case of fractional dimension, we only present the proof for the fractional case.

Proposition 3.4.1 [[FT00; BS92]] Let $\mathbf{P} = (X, P)$ be a poset which is not a linear order, and let $\mathbf{H_P}$ be its hypergraph of incomparable pairs. Then

a) $\mathbf{H_P}$ can be t:k-colored if and only if $\mathbf{P}$ has a k:t-realizer.

b) A linear extension L of $\mathbf{P}$ defines a vertex cover S by setting

$$S := \{(x, y) \in \text{inc}(\mathbf{P}) : (x, y) \in L\}$$

and an independent set I of $\mathbf{H_P}$ by setting

$$I := \{(x, y) \in \text{inc}(\mathbf{P}) : (y, x) \in L\}$$

Proof. For statement a), note that a t:k-coloring suggests a way of reversing each incomparable pair in at least k linear extensions, with each color corresponding to a linear extension. Since a valid coloring does not use a color common to all vertices of a hyperedge, these extensions do not contain any cycles and are therefore valid partial orders. On the other hand, a k:t-realizer suggests a way of coloring the vertices of the graph by assigning color i to each pair $(a, b) \in \text{inc}(\mathbf{P})$ if and only if it is reversed in L_i. Since a linear extension can only reverse a proper subset of vertices for each hyperedge, there is no hyperedge that contains vertices that share a common color. Therefore, the coloring is valid.

Statement b) follows from the above arguments. Since all pairs in the set I are assigned the same color in the above discussed coloring, they must form an independent set, by validity of the coloring. Observe that, since for each $(a, b) \in \text{inc}(P)$ a linear extension L must either include it or reverse it, S is the complement of I. As complements of independent sets are vertex covers, S forms a vertex cover of $\mathbf{H_P}$. □

3.5 The algorithmic approximation framework

In this section we bring together all previously discussed components in order to create a framework that yields better than 2-approximation algorithms for families of posets that have bounded fractional dimension.

The following observation is central to our approach. Recall that in Section 3.3.4 we presented several formulations and relaxations of the problem $1|\text{prec}|\sum_j w_j C_j$, resulting in the vertex cover problem in graph $G_{\mathbf{P}}^S$, associated to the given instance of the scheduling problem. It turns out that this graph reflects the structure of the poset given in the instance of $1|\text{prec}|\sum_j w_j C_j$ in such a way that we can apply techniques from Dimension Theory when trying to approximate the vertex cover problem. The following section makes this intuition explicit.

3.5.1 Structure of the Graph $G^S_{\mathbf{P}}$

Consider an instance S of $1|\text{prec}|\sum_j w_j C_j$ with precedence constraints $\mathbf{P} = (N, P)$. Recall that the vertices of $G^S_{\mathbf{P}}$ are the incomparable pairs of $\mathbf{P}$. Graph $G^S_{\mathbf{P}}$ has three types of edges (depicted in Figure 3.10), one for each type of constraint in [CS-IP] (See Section 3.3.2):

(i) Two vertices (i, j) and (j, i) are adjacent (constraints (3.3b)).

(ii) Two vertices (i, k) and (k, j) are adjacent if $(i, j) \in P$ (constraints (3.3c)).

(iii) Two vertices (i, ℓ) and (k, j) are adjacent if $(i, j) \in P$ and $(k, \ell) \in P$ (constraints (3.3d))

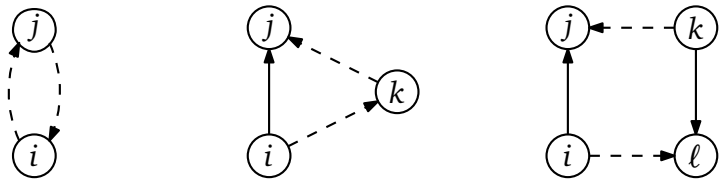


Figure 3.10. The three different edge types of graph $G^S_{\mathbf{P}}$, corresponding to the three constraints of [CS-IP]. A pair of jobs $i \| j$ is connected by a dashed line, while solid lines represent precedence constraints.

It is not hard to see that the edge set of $G^S_{\mathbf{P}}$ consists of those sets U of *two* incomparable pairs that cannot be reversed at the same time without introducing cycles. This observation is crucial to our approach, since it implies that the graph $G^S_{\mathbf{P}}$ and the graph of incomparable pairs of partial orders (see Section 3.4) coincide.

Proposition 3.5.1 The vertex cover graph $G^S_{\mathbf{P}}$ associated to $1|\text{prec}|\sum_j w_j C_j$ and the graph of incomparable pairs $G_{\mathbf{P}}$ coincide.

Theorem 3.5.2 ([Tro92; CS05]) *Let $\mathbf{P} = (N, P)$ be a poset that is not a linear order. Then the graph $G_{\mathbf{P}}$ is bipartite if and only if $dim(\mathbf{P}) = 2$.*

Note that the example poset given in Figure 3.6 is 2-dimensional. Indeed, the graph of incomparable pairs of this poset is a tree (see Figure 3.9a), and thus bipartite. Theorem 3.5.2 is a well-known result in dimension theory. Using a different approach, Correa & Schulz [CS05] rediscovered it for the vertex cover graph $G^S_{\mathbf{P}}$, independent of the connection pointed out by Proposition 3.5.1.

3.5.2 The Framework

Propositions 3.5.1 and 3.4.1 imply that graph $G^S_{\mathbf{P}}$ can be colored using $dim(\mathbf{P})$ colors and, given a realizer, this can be done in polynomial time (see the proof of Proposition 3.4.1). Combining this with Hochbaum's [Hoc83] $(2-2/t)$-approximation algorithm for the weighted vertex cover problem, whenever the vertex cover graph is t-colorable in polynomial time, yields the following theorem.

Theorem 3.5.3 *Problem* $1|\text{prec}|\sum_j w_j C_j$ *has a polynomial time* $(2-\frac{2}{t})$*-approximation algorithm, whenever precedence constraints are given by a* t*-realizer.*

Using the correspondence between fractional dimension and fractional coloring, this theorem can be generalized to the case that the *fractional* dimension of the poset is bounded. However, there is a technicality that needs to be overcome.

Assume that we are given a $k{:}t$-realizer for which the ratio t/k is some small constant, while k and t are exponential in the size of the groundset (number of jobs). Since this gives a fractional chromatic number with a small ratio, such a realizer suggests a fractional coloring that can indeed be used to provide a $(2-2/(t/k))$-approximation. Clearly, however, our above sketched algorithm cannot run in polynomial time, since it takes exponential time to even read-in the realizer.

Randomness comes to our avail. In fact, randomization will merely serve as a detour, in which we first show a randomized algorithm that has the above approximation ratio *in expectation*. We then show that for all our applications this algorithm can be derandomized using the method of conditional probabilities. The result is a fully deterministic algorithm that guarantees the above approximation ratio.

We say that a poset $\mathbf{P}$ admits an *efficiently samplable* $k{:}t$-realizer if there exists a randomized algorithm that, in time polynomial in the size of the ground set, returns any linear extension from a $k{:}t$-realizer $\mathscr{F} = \{L_1, L_2, \ldots, L_t\}$ of $\mathbf{P}$ with probability $1/t$. Using this definition we can now formulate the generalization of Theorem 3.5.3. We revisit the algorithmic framework already sketched in Section 3.2 as a proof of this theorem (see Figure 3.11 for an illustration).

Theorem 3.5.4 *Problem* $1|\text{prec}|\sum_j w_j C_j$ *has a randomized* $(2-\frac{2}{t/k})$*-approximation algorithm, whenever precedence constraints admit an efficiently samplable* $k{:}t$*-realizer.*

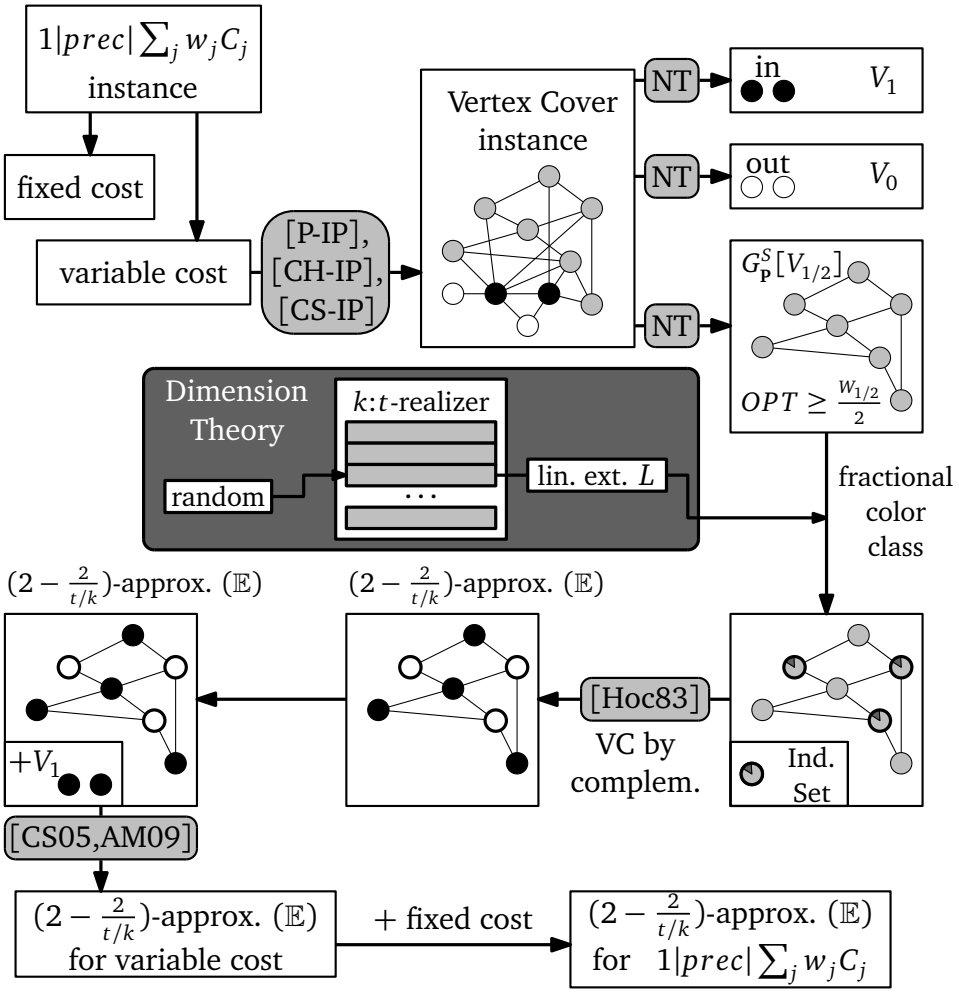


Figure 3.11. An illustration of the framework based on fractional coloring. The sign ($\mathbb{E}$) indicates that the claimed approximation ratios hold in expectation.

Proof. Let S be an instance of 1|prec|$\sum_j w_j C_j$ where precedence constraints are given by a poset $\mathbf{P} = (N, P)$ that admits an efficiently samplable $k{:}t$-realizer $\mathscr{F} = \{L_1, L_2, \ldots, L_t\}$. Furthermore, we assume that fdim($\mathbf{P}$) ≥ 2. The case when fdim($\mathbf{P}$) $= 1$, i.e. $\mathbf{P}$ is a linear order, is trivial.

Let $G^S_{\mathbf{P}} = (V_{\mathbf{P}}, E_{\mathbf{P}})$ be the weighted vertex cover instance associated to S where each vertex (incomparable pair) $v = (i, j) \in V_{\mathbf{P}}$ has weight $w_v = p_i \cdot w_j$, as specified in Section 3.3.2.

We apply the preprocessing given by Nemhauser & Trotter [NT73; NT75] described in Section 3.3.1. This outputs a partition of the vertices of the graph

into V_0, V_1 and $V_{1/2}$ such that only the induced subgraph $G_{\mathbf{P}}^S[V_{1/2}]$ needs to be further considered: adding the vertices V_1 yields a feasible solution that is an at least as good approximation as the one found in $G_{\mathbf{P}}^S[V_{1/2}]$.

To approximate vertex cover in $G_{\mathbf{P}}^S[V_{1/2}]$, we consider the linear extensions of $\mathscr{F}$ as outcomes in a uniform sample space. For an incomparable pair (x, y), the probability that $y > x$ in a linear extension picked from $\mathscr{F}$ is given by

$$\text{Prob}_{\mathscr{F}}[y > x] = \frac{\left|\{i = 1, \ldots, t : y > x \in L_i\}\right|}{t} \geq \frac{k}{t} \tag{3.4}$$

This inequality holds because every incomparable pair is reversed in at least k linear extensions of $\mathscr{F}$, by definition of a k:t-realizer. Let us pick one linear extension L uniformly at random from $\mathscr{F} = \{L_1, \ldots, L_t\}$. By Proposition 3.4.1, L defines the independent set

$$I_L := \left\{(x, y) \in \text{inc}(\mathbf{P}) : (y, x) \in L\right\},$$

i.e. the set of incomparable pairs that are reversed in L. By linearity of expectation, its expected weight equals[8]

$$\mathbb{E}[w(I_{1/2})] = \sum_{(i,j) \in V_{1/2}} Prob_{\mathscr{F}}[j > i] \cdot w_{(i,j)} \geq \frac{k}{t} \cdot W_{1/2}$$

A vertex cover solution C for the graph $G_{\mathbf{P}}^S[V_{1/2}]$ can be obtained by complementing I_L in $V_{1/2}$, i.e. setting $C = V_{1/2} \setminus I_{1/2}$. The expected value of this vertex cover solution is

$$\mathbb{E}[w(C)] = W_{1/2} - E[w(I_{1/2})] \leq \left(1 - \frac{k}{t}\right) W_{1/2}$$

We now add V_1 to C in order to get a vertex cover for the total graph $G_{\mathbf{P}}^S$. The expected value of this vertex cover solution is, by linearity of expectation,

$$\begin{aligned}
\mathbb{E}[w(V_1 \cup C)] &\leq W_1 + \left(1 - \frac{k}{t}\right) W_{1/2} && (3.5)\\
&\leq 2\left(1 - \frac{k}{t}\right)\left(W_1 + \frac{1}{2} W_{1/2}\right) && (3.6)\\
&\leq \left(2 - \frac{2}{t/k}\right) OPT && (3.7)
\end{aligned}$$

[8] Recall that W_i denotes $w(V_i)$ for $i \in \{0, 1/2, 1\}$

where inequality 3.6 follows from the fact that $2(1 - k/t) \geq 1$ because $t/k \geq \text{fdim}(\mathbf{P}) \geq 2$, and inequality 3.7 holds since $W_1 + 1/2 \cdot W_{1/2}$ is the optimal value of [VC-LP] and thus a lower bound on the actual optimum.

Finally, theorem 3.3.4 implies that any α-approximation algorithm for $G_{\mathbf{P}}^S$ also gives an α-approximation algorithm for S. Thus we obtain a randomized $(2 - \frac{2}{t/k})$-approximation algorithm for S. □

We point out that, since the dimension is an upper bound on the fractional dimension, the approach using the fractional dimension subsumes the approach using the (integral) dimension. Furthermore, it is worth noting that this framework improves or reaches the approximation ratios for all previously considered special cases of precedence constraints, to the best of our knowledge. Thus, it provides a unified way of generating the best known approximation algorithms for special cases of precedence constraints. We show some of these applications in the next section.

3.6 Applications of the framework

In this section we present applications of the framework presented in Section 3.5 for several families of precedence constraints. For each family, we first give a definition and discuss some properties of the poset, then give either a polynomial time algorithm that produces a realizer, or a polynomial time algorithm that samples a linear extension of a k:t-realizer uniformly at random. Once this is done, the approximation ratio claimed follows from the framework (see Section 3.5).

3.6.1 Interval Orders

A poset $\mathbf{P} = (N, P)$ is an *interval order* if it has an interval representation. An *interval representation* F for poset $\mathbf{P} = (N, P)$ assigns to each $x \in N$ a closed interval $I_x = [\ell_x, u_x] \subseteq \mathscr{R}$, so that $u_x < \ell_y$ in $\mathbb{R}$ if and only if $x < y$ in P (see Figure 3.12 for an example).

Interval orders allow for a nice characterization by minors (see e.g. [Tro92]): a poset is an interval order if and only if it does not contain the poset $\mathbf{2} + \mathbf{2}$ as a subposet (see Figure 3.12c). It is intuitive that such posets are not interval orders, because if I_a were completely before I_b and I_c were intersecting both, then any I_d that is completely after I_c would also have to be completely after I_a, but such an edge (a, d) is not present in the poset $\mathbf{2} + \mathbf{2}$.

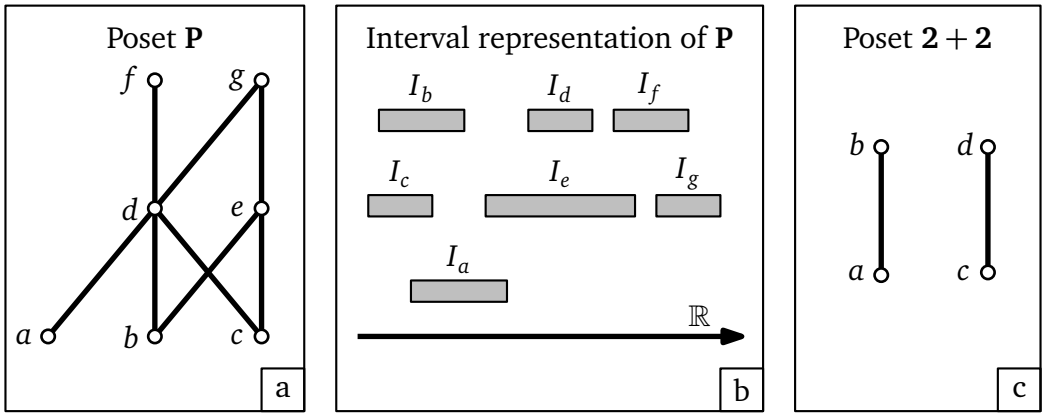


Figure 3.12. An example of an interval order (a), an interval representation for it (b) and the forbidden minor poset $\mathbf{2}+\mathbf{2}$ (c).

Interval orders can be recognized in $O(n^2)$ time [PY79]. The dimension of interval orders can be of order $\log\log n$ [Tro92], whereas the fractional dimension is known to be less than 4 [BS92], and this bound is asymptotically tight [FT94]. In the following we show how to obtain a 1.5-approximation algorithm for $1|prec|\sum_j w_j C_j$ with precedence constraints in the form of an interval order. By Theorem 3.5.4, it is sufficient to prove that interval orders admit an efficiently samplable $k{:}t$-realizer with $t/k \leq 4$.

Given a poset $\mathbf{P} = (N, P)$, disjoint subsets A and B of the ground set N, and a linear extension L of P, we say that B *is over* A *in* L if, for every incomparable pair of elements (a, b) with $a \in A$ and $b \in B$, one has $b > a$ in L. The following property of interval orders is fundamental for our approach.

Theorem 3.6.1 (Rabinovitch [Rab78]) *A poset* $\mathbf{P} = (N, P)$ *is an interval order if and only if for every pair* (A, B) *of disjoint subsets of* N *there is a linear extension* L *of* P *with* B *over* A.

We remark that given a pair (A, B) of disjoint subsets of N, we can find a linear extension L of P with B over A in polynomial time: add the relations $A \times B$ to P and complete the partial order to obtain a linear extension L. Using this property we can easily obtain a $k{:}t$-realizer $\mathscr{F} = \{L_1, \ldots, L_t\}$ with $t = 2^n$ and $k = 2^{n-2}$, where $n = |N|$, as follows.

Consider every subset A of N and let L_A be a linear extension of P in which $B = N \setminus A$ is over A. Now let $\mathscr{F}$ be the multiset of all the L_A's. Note that $|\mathscr{F}| = 2^n$. Moreover, for any incomparable pair (x, y) there are at least $k = 2^{n-2}$ linear extensions in $\mathscr{F}$ in which this pair is reversed. This is because, for each partition

(A, B) in which $x \in B$ and $y \in A$, this pair is reversed, and there are 2^{n-2} such partitions.

As mentioned in Section 3.5.2, this $k{:}t$-realizer is not helpful by itself, due to its exponential size. However, we can easily show that it is efficiently samplable: for every element (job) in the groundset, put it in A or in B with the same probability $1/2$.

By the previous observations and Theorem 3.5.4, we have a randomized polynomial time 1.5-approximation for $1|prec|\sum_j w_j C_j$ with interval order precedence constraints. In Section 3.6.1 we show that this algorithm can easily be derandomized using the standard method of conditional probabilities. This yields the following theorem.

Theorem 3.6.2 *Problem* $1|\text{prec}|\sum_j w_j C_j$ *for which the precedence constraints form an interval order has a 1.5-approximation algorithm.*

Derandomization for Interval Orders

Our goal is to partition the set of jobs into two sets A and B, such that any linear extension L with B over A will define an independent set with weight at least $W_{1/2}/4$. We will define such a partition incrementally.

Fix a subset of the jobs $N_i \subseteq N$ and a partition (A_i, B_i) thereof. Consider the set of all linear extensions of P that put B_i over A_i, and any given pair $(x, y) \in \text{inc}(P)$. The following cases can be distinguished:

- If $x \in A_i, y \in B_i$ then this pair is reversed in all such linear extensions.
- If $x \in A_i, y \in \overline{A_i \cup B_i}$ then (x, y) is certain to be reversed[9] only in case of the event that $y \in B$ when the sets A,B have been fully defined. Similarly, when $y \in B_i, x \in \overline{A_i \cup B_i}$, (x, y) is certain to be reversed in the event that $x \in A$.
- If $x, y \in \overline{A_i \cup B_i}$ then both events $x \in A$ and $y \in B$ need to occur in order for the pair (x, y) to be reversed with certainty in a linear extension.

Consider a function Φ defined as follows:

$$\Phi(A_i, B_i) = \sum_{(x,y)\in A_i \times B_i} p_y w_x + \sum_{\substack{x\in A_i, y\notin A_i\cup B_i \text{ or} \\ x\notin A_i\cup B_i, y\in B_i}} \frac{p_y w_x}{2} + \sum_{x,y\notin A_i\cup B_i} \frac{p_y w_x}{4}$$

[9] Note that such a pair might be reversed also when both x, y are in the same set A or B, but this cannot be concluded with certainty

Note that $\Phi(A_i, B_i)$ gives a lower bound on the expected value of the independent set conditioned upon our current choices of A_i and B_i.

Set $A_0 = B_0 = \emptyset$ and observe that $\Phi(A_0, B_0) = W_{1/2}/4$. For $i = 1, \ldots n$ we have to decide if job i is in set A_i or in set B_i. We evaluate both possibilities:

1. $A_i^1 := A_{i-1} \cup \{i\}$ and $B_i^1 := B_{i-1}$
2. $A_i^2 := A_{i-1}$ and $B_i^2 := B_{i-1} \cup \{i\}$

Let $g = \arg\max_{h=1,2}\{\Phi(A_i^h, B_i^h)\}$ and observe that

$$\Phi(A_{i-1}, B_{i-1}) \leq \Phi(A_i^g, B_i^g).$$

We therefore set

$$\begin{aligned} A_i &:= A_i^g \\ B_i &:= B_i^g. \end{aligned}$$

At the end we have partitioned the set of jobs into two sets A_n and B_n such that $\Phi(A_n, B_n) \geq w(V_{1/2})/4$. Since P is an interval order $P \cup \{(a, b) \in A_n \times B_n : a||b\}$ is a valid extension of P and any linear extension of it gives an independent set of value $\geq w(V_{1/2})/4$.

3.6.2 Semiorders

A special case of interval orders are semiorders. An interval order $\mathbf{P} = (N, P)$ is called a *semiorder* (or unit interval order) if $\mathbf{P}$ has an interval representation in which all intervals have the same length. More precisely, an interval order is a semiorder if there is a representation assigning to each $x \in N$ an interval $I_x = [a_x, a_x + 1]$ so that $a_x + 1 < a_y$ if and only if $x < y$ in P.

Just as interval orders, semiorders allow for a characterization by forbidden minors as well. Besides the minor poset $\mathbf{2} + \mathbf{2}$ inherited from interval orders, semiorders have also $\mathbf{3} + \mathbf{1}$ as a forbidden minor, depicted in Figure 3.13. Intuitively, such a poset admits a representation by intervals only if the interval I_b is strictly contained in the interval I_d, which is impossible due to the unit length of intervals. Observe, for example, that the interval order of Figure 3.12 is not a semiorder, since subposet induced by the elements $\{a, d, e, f\}$ is a $\mathbf{3} + \mathbf{1}$ poset.

Semiorders can be recognized in $O(n^2)$ time (see e.g. [Möh89; Tro92]). In contrast to interval orders that have unbounded dimension [Tro92], Rabinovitch proved, by constructing a realizer, that the dimension of semiorders is at most

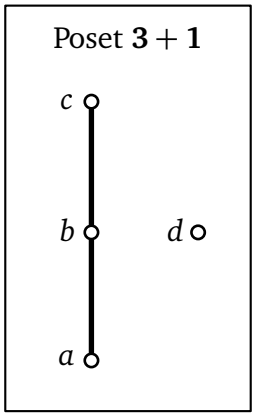


Figure 3.13. The additional forbidden subposet $\mathbf{3}+\mathbf{1}$ for semiorders.

three [Rab78] (see also [Tro92] for a good explanation). The result presented in this section is achieved using the approach based on (integral) dimension.

The realizer proposed by Rabinovitch is constructed as follows (see Figure 3.14 for an overview). Intuitively, the elements of the groundset will be partitioned into "levels" by iterative removal of the maximal elements of the residual poset, where the maximal elements of a poset $\mathbf{P}$ are

$$\max(\mathbf{P}) := \{x \in N : \forall y \in N, \text{ either } (y,x) \in P \text{ or } (y,x) \in \text{inc}(\mathbf{P})\}.$$

Those levels will be used to define two sets A and B, one containing the odd levels and the other one containing the even levels. Two linear extensions are formed, one putting A over B and the other B over A, in order to reverse all incomparable pairs that span across levels. A third linear extension is used to reverse incomparable pairs within levels and completes the definition of the realizer. A more precise description of this algorithm follows.

Given a semiorder $\mathbf{P} = (N,P)$, we iteratively define $N_i, P_i, \mathbf{P}_i$ and A_i for $i \leq i \leq h$ (for some[10] h) as follows: let $N_1 = N, P_1 = P, \mathbf{P}_1 = (N_1,P_1)$ and let $A_1 = \max(\mathbf{P}_1)$, i.e. the maximal elements of the poset $\mathbf{P}_1$. If $N_i, P_i, \mathbf{P}_i$ and A_i have been defined for some $i : 1 \leq i < h$, set $N_{i+1} = N_i \setminus A_i$, $P_{i+1} = P_i \cap (N_{i+1} \times N_{i+1})$, $\mathbf{P}_{i+1} = (N_{i+1}, P_{i+1})$, and $A_{i+1} = \max(\mathbf{P}_{i+1})$. Note that the sets $A_1, A_2, \ldots, A_h$ form a partition of N and all elements in a set A_i are incomparable[11]. Let $A = \{x \in N : x \in A_i \text{ for some odd } i\}$ and let $B = N \setminus A$. As semiorders are a special case of interval orders, we can construct two linear extensions L_1 and L_2 so that B

[10] This integer is called the *height* of the poset

[11] They are *antichains*, in Dimension Theory terminology

is over A in L_1 and A is over B in L_2 (see Theorem 3.6.1). Finally, we construct a third linear extension L_3 so that an incomparable pair $(i,j) \in L_3$, if either (i) $i \in A_k$ and $j \in A_\ell$ with $k > \ell$ or (ii) $\{i,j\} \subseteq A_k$ and $(j,i) \in L_1$. $\mathscr{R} := \{L_1, L_2, L_3\}$ is a realizer for $\mathbf{P}$, since incomparable pairs that span across different A_i's are reversed in either L_1 or L_2, while incomparable pairs within some A_i are reversed either in L_1 or in L_3 (see [Rab78; Tro92] for more details).

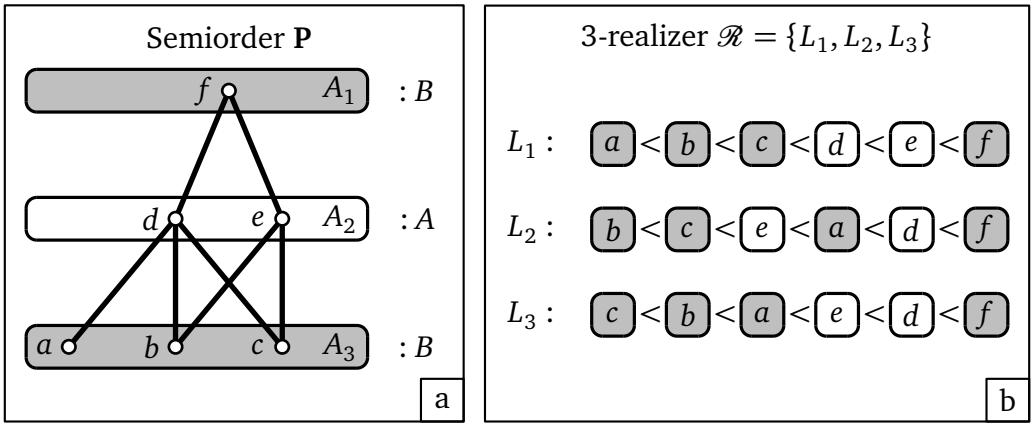


Figure 3.14. Overview of the construction of a 3-realizer for semi-orders. The jobs are partitioned into two sets A and B, in white and gray respectively. L_1 puts B/A, L_2 puts A/B and L_3 reverses incomparable pairs within a set A, B that are not reversed in L_1.

It is not hard to see that we can construct $\mathscr{R}$ in polynomial time. This together with Theorem 3.5.3, gives us the following theorem.

Theorem 3.6.3 *Problem* $1|\text{prec}|\sum_j w_j C_j$ *for which the precedence constraints form a semiorder has a* $(1+1/3)$*-approximation algorithm.*

3.6.3 Orders of Interval Dimension two

The *interval dimension* of a poset $\mathbf{P} = (N, P)$, denoted by $\dim_I(\mathbf{P})$, is defined [Tro92] as the least t for which there exist t extensions $Q_1, Q_2, \ldots, Q_t$, so that:

- $P = Q_1 \cap Q_2 \cap \cdots \cap Q_t$ and
- (N, Q_i) is an interval order for $i = 1, 2, \ldots, t$.

Obviously, if $\mathbf{P}$ is an interval order $\dim_I(\mathbf{P}) = 1$. Moreover, since any linear extension is in particular an interval order but not vice versa, $\dim_I(\mathbf{P}) \leq \dim(\mathbf{P})$.

The class of posets of interval dimension 2 forms a proper superclass of the class of interval orders. Posets of interval dimension two can be recognized in $O(n^2)$ time due to Ma & Spinrad [MS94]. Given a poset $\mathbf{P}$ with $\dim_I(\mathbf{P}) = 2$, their algorithm also yields an interval realizer $\{Q_1, Q_2\}$. As described in Section 3.6.1, we know how to construct k:t-realizers $\mathscr{F}_1$ and $\mathscr{F}_2$ for the interval orders Q_1 and Q_2 respectively, with $k = 2^{n-2}$,$t = 2^n$. Forming the union of these realizers, we obtain a k:t-realizer $\mathscr{F} := \mathscr{F}_1 \cup \mathscr{F}_2$ with $t = 2^{n+1}$ and $k = 2^{n-2}$. Note that in $\mathscr{F}$ each incomparable pair is reversed in at least $1/8$ of the linear extensions. Furthermore, we can efficiently pick uniformly at random one linear extension from $\mathscr{F}$: pick uniformly at random a linear extension from either $\mathscr{F}_1$ or $\mathscr{F}_2$ with the same probability $1/2$. The derandomization is analogous to the one for interval orders presented in Section 3.6.1 and is omitted. This yields the following theorem.

Theorem 3.6.4 *Problem* $1|\text{prec}|\sum_j w_j C_j$*, whenever precedence constraints have interval dimension at most* 2*, has a* 1.75*-approximation algorithm.*

3.6.4 Posets of Bounded Up- or Down-degree

In the following we will see how to obtain, using Theorem 3.5.4, an approximation algorithm for $1|\text{prec}|\sum_j w_j C_j$ when the precedence constraints form a poset whose up-degree (or down-degree) is bounded. We first give some definitions.

Let $\mathbf{P} = (N, P)$ be a poset. For any job $j \in N$, define the *degree of* j, denoted $\deg(j)$, as the number of jobs comparable (but not equal) to j in $\mathbf{P}$. Let $\Delta(\mathbf{P}) = \max\{\deg(j) : j \in N\}$. Given a job j, $D(j)$, called the *down-set of* j *in* $\mathbf{P}$, denotes the set of all jobs which are less than j. Similarly, $U(j)$ is called the *up-set of* j *in* $\mathbf{P}$ and denotes those jobs which are greater than j in P. We call $\deg_D(j) = |D(j)|$ the *down-degree* of job j and let $\Delta_D(\mathbf{P}) = \max\{\deg_D(j) : j \in N\}$. The *up-degree* $\deg_U(j)$ and $\Delta_U(\mathbf{P})$ are defined analogously.

We observe that the NP-completeness proof for $1|\text{prec}|\sum_j w_j C_j$ given by Lawler [Law78] was actually provided for posets $\mathbf{P}$ with $\Delta_D(\mathbf{P}) = 2$. Using fractional dimension we show that these posets (with bounded $\min\{\Delta_D, \Delta_U\}$) allow for a better than 2-approximation.

Theorem 3.6.5 *Problem* $1|\text{prec}|\sum_j w_j C_j$ *has a polynomial time* $(2-2/f)$*-approximation algorithm, where* $f = 1 + \min\{\Delta_D, \Delta_U, 1\}$.

Proof. Let $\mathbf{P} = (N, P)$ be the poset representing the precedence constraints with bounded $\min\{\Delta_D, \Delta_U\}$. We will show that $\mathbf{P}$ has an efficiently samplable k:t-realizer with $t/k \leq \min\{\Delta_D, \Delta_U\} + 1$ using a result by Felsner & Trotter [FT94].

To describe their approach we need to first introduce some concepts. Assume, without loss of generality, that **P** is not decomposable with respect to lexicographic sums (see Section 3.6.5)[12]. Otherwise, a decomposition with respect to lexicographic sums can be done in $O(n^2)$ time (see e.g. [Möh89]), and each component will have degree no larger than the degree of **P** and can be considered separately (see Theorem 3.6.6). We call an incomparable pair $(x, y) \in \text{inc}(\mathbf{P})$ a *critical pair* if for all $z, w \in N \setminus \{x, y\}$

1. $z < x$ in P implies $z < y$ in P, and
2. $y < w$ in P implies $x < w$ in P.

Critical pairs play an important role in dimension theory: any incomparable pair (a, b) can be associated with at least one critical pair (x, y) so that a linear extension reversing (x, y) also reverses (a, b) [Tro92]. It follows that if for each critical pair (x, y), there are at least k linear extensions in $\mathscr{F} = \{L_1, \ldots, L_t\}$ which reverse the pair (x, y) then $\mathscr{F}$ is a $k{:}t$-realizer of P and vice versa [BS92].

For an element $x \in N$ and a set $A \subseteq N$ we write $x > A$, if $x > y$ for all $y \in A$. For any permutation M of N, consider the set $C(M)$ of critical pairs (x, y) that satisfy the following two conditions:

1. $x > (D(y) \cup \{y\})$ in M if $|D(y)| < \Delta_D$
2. $x > D(y)$ in M if $|D(y)| = \Delta_D$

In [FT94], Felsner & Trotter present an algorithm (for posets that are not decomposable with respect to lexicographic sums) that converts in polynomial time a permutation M of N into a linear extension L of P so that L reverses all critical pairs in the set $C(M)$. Now set $t = |N|!$ and consider the set $\mathscr{M} = \{M_1, M_2, \ldots, M_t\}$ of all permutations of the ground set N. Observe that for any critical pair (x, y) there are at least $n!/(\Delta_D + 1)$ different permutations $M_i \in \mathscr{M}$, where the critical pair is reversed, i.e. $(y, x) \in C(M_i)$. This is because any ordering of the n elements defines an ordering of $T := \{x\} \cup D(y)$ (respectively, an ordering of $T := \{x\} \cup \{y\} \cup D(y)$) and in $1/|T| \geq 1/(\Delta_D + 1)$ of them x is the last element of T.

Applying the algorithm in [FT94] we obtain a $k{:}t$-realizer $\mathscr{F} = \{L_1, \ldots, L_t\}$ of P with $t = n!$ and $k = n!/(\Delta_D + 1)$. Moreover, we can efficiently pick uniformly at random one linear extension from $\mathscr{F}$: generate uniformly at random one permutation of jobs (e.g. by Knuth's shuffle algorithm [Knu69]) and transform it into a linear extension with the described properties using the algorithm in [FT94]. As

[12] This is needed in order to be able to apply the algorithm of Felsner & Trotter [FT94] below.

described below, the algorithm can be derandomized using the standard method of conditional probabilities. Finally observe that we can repeat a similar analysis using Δ_U instead of Δ_D: let $\mathbf{P}^d = (N, P^d)$, where $P^d = \{(i,j) : (j,i) \in P\}$ then $\Delta_D(P^d) = \Delta_U(P)$ and a k:t-realizer $\mathscr{R}^d = \{L_1^d, \ldots, L_t^d\}$ of $\mathbf{P}^d$ gives a k:t-realizer $\mathscr{R} = \{L_1, \ldots, L_t\}$ of $\mathbf{P}$, where $L_i = \{(i,j) : (j,i) \in L_i^d\}$. □

We remark that it is necessary to use *fractional* dimension for obtaining the above result. To see this, consider the *incidence poset* $\mathbf{P}(G) = (N, P)$ defined as follows: given an undirected graph $G(V,E)$, let $N = V \cup E$ and for every $v \in V$ and $e = \{v_1, v_2\} \in E$, put $(v,e) \in P$ if and only if $v \in \{v_1, v_2\}$. Since every edge is adjacent to only two vertices, Δ_D is bounded by 2. For K_n the complete graph on n nodes, Spencer [Spe71] showed that $\dim(\mathbf{P}(K_n)) = \Theta(\log\log n)$ whereas from the above discussion we have $\mathrm{fdim}(\mathbf{P}(K_n)) \leq 1 + \min\{\Delta_U, \Delta_D\} = 3$.

Derandomization for Bounded Degree Posets

We let $V_{1/2}$ be the set of vertices with value $1/2$ in the optimal solution to the [VC-LP] formulation of the scheduling problem (see Section 3.5.2).

We consider the case when $\Delta_D \leq \Delta_U$. The case when $\Delta_U < \Delta_D$ is symmetric and omitted. It suffices to compute a permutation that gives a linear extension whose associated independent set has value at least

$$\frac{w(V_{1/2})}{\Delta_D + 1}.$$

We already mentioned that any incomparable pair (a,b) can be associated with at least one critical pair (x,y) so that a linear extension reversing (x,y) also reverses (a,b) [Tro92]. For simplicity, we associate every incomparable pair with *exactly* one critical pair such that the above condition holds and we denote by $C_{(x,y)}$ the set of incomparable pairs associated to the critical pair (x,y). Note that by convention we have $(x,y) \in C_{(x,y)}$ and the set $\{C_{(x,y)} : (x,y) \text{ is a critical pair}\}$ forms a partition of the incomparable pairs. From the proof of Theorem 3.6.5, it follows that the probability that any critical pair (x,y) (and thus the incomparable pairs in $C_{(x,y)}$) are reversed by a linear extension L obtained from a uniformly picked permutation is at least

$$N_{(x,y)}/D_{(x,y)},$$

where at the beginning

$$N_{(x,y)} := 1 \text{ and } D_{(x,y)} := \begin{cases} |D(y)| + 2, & \text{if } |D(y)| < \Delta_D \\ |D(y)| + 1, & \text{if } |D(y)| = \Delta_D \end{cases}.$$

Starting from the first position of the permutation, we consider the n possibilities corresponding to placing any job at that position and retain the best one. Then we remove the job that has been placed at the first position and continue with the second position, by considering the remaining $n-1$ jobs, and so forth until the end of the permutation. Each time we consider a possibility, we update the probabilities accordingly. For example, if we decide to put job j at position i then the probability to reverse the incomparable pairs in $C_{(x,y)}$ is updated as follows.

- Set $N_{(x,y)} := 0$ if $x = j$, $|D(y)| < \Delta_D$ and y has not been placed at a previous location;
- Set $N_{(x,y)} := 0$ if $x = j$ and there exists a $z \in D(y)$ that has not been placed at a previous location;
- Set $D_{(x,y)} := D_{(x,y)} - 1$, if ($j = y$ and $D(y) < \Delta_D$) or $j \in D(y)$.

With the updated probabilities we can compute the associated value and retain the best choice.

3.6.5 Lexicographic Sums

In this section, we show how the results of the previous sections can be combined in order to provide approximation algorithms for a wider range of posets. The construction we will use is called the *lexicographic sum* of posets (see e.g. [Tro92]) and derives from the following simple idea. Take a poset $\mathbf{P} = (N, P)$ and replace each of its elements $x \in N$ with a partially ordered set $\mathbf{Q_x}$ such that each element of $\mathbf{Q_x}$ has the same relation to points outside it as x had before the replacement (See Figure 3.15). A more formal definition follows.

For a poset $\mathbf{P} = (N, P)$ and a family of posets $\mathscr{S} = \{(Y_x, Q_x) \mid x \in N\}$ indexed by the elements in N, the lexicographic sum of $\mathscr{S}$ over (N, P), denoted $\sum_{x\in(N,P)}(Y_x, Q_x)$ is the poset (Z, R) where $Z = \{(x, y) \mid x \in N, y \in Y_x\}$ and $(x_1, y_1) \leq (x_2, y_2)$ in R if and only if one of the following two statements holds:

1. $x_1 < x_2$ in P.
2. $x_1 = x_2$ and $y_1 \leq y_2$ in Q_{x_1}.

We call $\mathscr{P} = \{\mathbf{P}\} \cup \mathscr{S}$ the *components* of the lexicographic sum. A lexicographic sum is *trivial* if $|N| = 1$ or if $|Y_x| = 1$ for all $x \in N$. A poset is *decomposable with respect to lexicographic sums* if it is isomorphic to a non-trivial

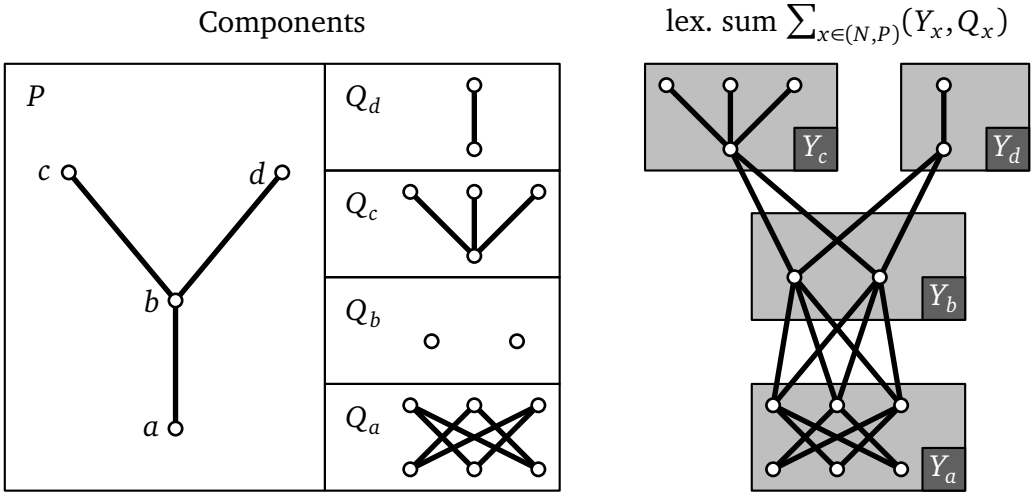


Figure 3.15. An example of the lexicographic sum $\sum_{x\in(N,P)}(Y_x, Q_x)$ of the posets Q_a, Q_b, Q_c and Q_d over P.

lexicographic sum. Moreover, a decomposition of a poset with respect to lexicographic sums can be done in $O(n^2)$ time (see e.g. [Möh89])

In case the precedence constraints of every component admit an efficiently samplable realizer, we observe that this translates into a randomized approximation algorithm:

Theorem 3.6.6 *Problem* $1|\text{prec}|\sum_j w_j C_j$*, whenever precedence constraints form a lexicographic sum whose components admit efficiently samplable* $k{:}t$*-realizers, has a randomized* $(2 - \frac{2}{t/k})$*-approximation algorithm.*

Proof. Let $\mathbf{P} = (N, P)$ with $N = \{1, 2, \ldots, n\}$ and $\mathbf{Q}_1, \mathbf{Q}_2, \ldots, \mathbf{Q}_n$ be the components of the lexicographic sum $\sum_{x\in(N,P)} \mathbf{Q}_x$, where $\mathbf{Q}_x = (Y_x, Q_x)$.

Given a linear extension L_p of P and linear extensions $L_1, L_2, \ldots, L_n$ of the components $Q_1, Q_2, \ldots, Q_n$, we can construct a linear extension of $\sum_{x\in(N,P)} \mathbf{Q}_x$ by adding the relation $(x, y) \leq (x', y')$ between two incomparable elements of $\sum_{x\in(N,P)} \mathbf{Q}_x$, if either (i) $x < x'$ in L_p or (ii) $x = x'$ and $y < y'$ in L_x.

Now, suppose that all components have efficiently samplable $k{:}t$-realizers. Then we can sample each $k{:}t$-realizer independently to obtain linear extensions of all components, which in turn define a linear extension L of the lexicographic sum $\sum_{x\in(N,P)} \mathbf{Q}_x$. It is not hard to see that each incomparable pair of $\sum_{x\in(N,P)} \mathbf{Q}_x$ is reversed in L with probability at least k/t. Hence, the lexicographic sum has an efficiently samplable $k{:}t$-realizer. This together with Theorem 3.5.4 concludes the proof. □

Finally, we point out that, if the approximation algorithm for each component can be derandomized, this yields a deterministic approximation algorithm for the lexicographic sum. In particular this can be done when all components have low dimension.

3.7 Conclusions and future research

Even though our framework improved the previously considered special cases of precedence constraints, it failed to yield a better than 2-approximation for the general case. Understanding the exact approximability of the scheduling problem remains one of the most prominent open questions in scheduling theory. Furthermore, it would be interesting to investigate if the discovered connection between precedence constraint scheduling and graph theory can be further exploited, in order to devise even stronger approximation algorithms for the special cases considered. In particular, recent developments such as semidefinite programming [GW95] suggest themselves as alternatives to the currently used graph coloring approach. It would be interesting to investigate if this would lead to a stronger framework.

It is natural to expect new insights for the scheduling problem from a better characterization and understanding of the structure of the graph of incomparable pairs. For instance, the solvability of 2-dimensional precedence constraint scheduling can be gazed from the fact that the resulting graph of incomparable pairs is bipartite. Furthermore, we know that the graph of incomparable pairs is complete if and only if the precedence constraints are the standard examples of partial orders. However, a deeper understanding of the structural properties of this graph eludes us. We believe that useful insights for the scheduling problem will emerge from such a study.

Chapter 4

Single Machine Scheduling with Scenarios

In this chapter, we will discuss a robust variant of the problem $1 \mid\mid \sum_j w_j C_j$. In Section 2.1 we saw that this problem can be solved in polynomial time, simply by balancing the trade-offs between processing times and weights (see Theorem 2.1.3). However, the solution produced in this way may be far from optimality, if the numerical parameters of the instance change. Thus, such a solution is only of limited use when the instance cannot be formulated in a precise way. Instead, a new approach is needed that takes into account possible fluctuations of the parameters and produces a solution that is robust, i.e. whose performance doesn't deteriorate dramatically in case the real values of the parameters turn out to be different to the ones given by the input.

In the following, we will suggest a way to formulate the problem $1 \mid\mid \sum_j w_j C_j$ in an uncertain environment.

4.1 Deterministic vs. Robust Optimization

For many applications, the exact formulation of an instance of a problem is not always possible. The field of robust approximation deals with problems that are considered under the light of uncertainty. Since the optimality of a solution to a combinatorial optimization problem can change dramatically if the numerical parameters are perturbed, a new approach is needed to deal with decision-making in uncertain environments.

Two common approaches to deal with uncertainty are *stochastic optimization* (see e.g. [BL97]) and *robust optimization* (see e.g. [BS02; KY97]). In the stochastic optimization approach, the numerical values are assumed to be drawn

from some density function and the goal is to optimize the *expected value* of the objective function. On the other hand, the *robust optimization* can be considered the "worst-case counterpart" of the stochastic optimization approach. In this approach, the configuration of the numerical parameters is assumed to be drawn from a (potentially infinite) set of possible configurations. The goal is to construct a solution whose objective function value in the worst case scenario is as good as possible. Naturally, which scenario is the worst case scenario will in general depend on the solution at hand. This makes the combinatorics of such problems significantly more elaborate than their underlying fixed-parameters problems. Indeed, we will see in Section 4.4.1 that strong hardness results can be obtained for the robust version of problem $1|\text{prec}|\sum_j w_j C_j$, when the number of scenarios is not bounded by any constant. Obtaining strong inapproximability results for the classical version of problem $1|\text{prec}|\sum_j w_j C_j$ (i.e. in the absence of uncertainty) remains a challenging open question to this date.

4.1.1 Robustness Criteria and Uncertainty Modeling

Consider a combinatorial minimization[1] problem Π and let S be the (possibly infinite) set of all possibly realizable input data scenarios. Furthermore, let X be the set of the decision variables in the definition of the problem and D be the set of input data. The instance of the input data corresponding to scenario s is denoted by D^s, whereas the set of corresponding feasible solutions is denoted by F^s. The definition of the problem instance is completed by giving an objective function $\text{val}(\cdot)$ that maps a tuple (X, D^s) to the objective value $\text{val}(X, D^s)$. The optimal single scenario decision X_s^* for the input data instance D^s is the solution to a classical, deterministic optimization problem and satisfies

$$z^s = \text{val}(X_s^*, D^s) = \min_{X \in F^s}(\text{val}(X, D^s))$$

Kouvelis & Yu [KY97] introduced the following three robustness criteria:

the absolute robust decision X_A is defined as the one that minimizes the maximum total cost, among all feasible decisions over all realizable input data scenarios, i.e.

$$z_A = \max_{s \in S} \text{val}(X_A, D^s) = \min_{X \in \bigcap_{s \in S} F_s} \max_{s \in S} \text{val}(X, D^s).$$

This optimization criterion is often referred to as the "MIN-MAX" version of the problem Π.

[1] The definitions for maximization problems are analogous

the robust deviation decision X_D is defined as the one that exhibits the best worst case deviation from optimality among all feasible decisions over all realizable input data scenarios, i.e.

$$z_D = \max_{s \in S}(\text{val}(X_d, D^s) - \text{val}(X_s^*, D^s)) = \min_{X \in \bigcap_{s \in S} F_s} \max_{s \in S}(\text{val}(X, D^s) - \text{val}(X_s^*, D^s)).$$

This optimization criterion is often referred to as the "MIN-MAX REGRET" version of the problem Π. Recent examples of these two families of approaches can be found in [ABV06; KZ07; FJMM07].

the relative robust decision X_R is defined as the one that exhibits the best worst case percentage deviation from optimality, among all feasible decisions over all realizable input data scenarios, i.e.

$$z_R = \max_{s \in S} \frac{\text{val}(X_R, D^s) - f(X_s^*, D^s)}{\text{val}(X_s^*, D^s)} = \min_{X \in \bigcap_{s \in S} F_s} \max_{s \in S} \frac{\text{val}(X, D^s) - \text{val}(X_s^*, D^s)}{f(X_s^*, D^s)}.$$

As noted by Kouvelis & Yu [KY97], this criterion is often given in its equivalent form

$$z_R = \min_{X \in \bigcap_{s \in S} F_s} \max_{s \in S} \frac{\text{val}(X, D^s)}{f(X_s^*, D^s)}$$

since the remaining part of the formula is constant.

We note that the above criteria definitions get simplified when the feasibility of the solution is independent of the input data scenario, i.e. $\bigcap_{s \in S} F_s = F_s$ for all $s \in S$. This is, for example, the case for some ordering problems, including the robust scheduling problem discussed in Section 4.2. Moreover, for ease of notation we will write $\text{val}(X, s_\ell)$ instead of $\text{val}(X, D^{s_\ell})$ in the following.

There are two main approaches in the definition of scenarios. In the *discrete scenario* case, the definition of the instance contains a finite set S of input data scenarios s, each of which is represented by a vector that defines the values D^s. The robust version of a problem Π in this manner is often referred to in the literature as "Π with scenarios". In the *interval data* case, the definition of the instance contains a real interval I_d for each of the input data d. The infinite set of realizable input data scenarios is implicitly defined as the set of all vectors defining the values D^s, such that $d \in I_d$.

4.1.2 Approximability of Robust Problems

In a recent paper, Aissi et. al [ABV06] considered the approximability of min-max versions of several combinatorial optimization problems. Their approach

to model uncertainty was to use discrete scenarios. Their results contain robust versions of several basic combinatorial problems, namely the MIN-MAX KNAPSACK problem, the MIN-MAX SHORTEST PATH problem, and the MIN-MAX SPANNING TREE problem. More specifically, when the number of scenarios is bounded, the authors proved the existence of FPTAS's for both the MIN-MAX and the MIN-MAX REGRET versions of these problems, with the exception of the MIN-MAX REGRET KNAPSACK problem, which was shown not to be approximable at all. For an unbounded number of scenarios, the MIN-MAX SHORTEST PATH and MIN-MAX SPANNING TREE were shown not to be approximable within a specified constant, while the MIN-MAX KNAPSACK problem was shown not to be approximable at all. For the unbounded scenario case, the behaviour of the problem with regards to approximability was retained when the relative robustness criterion instead of the absolute robustness criterion was considered.

Recently, Kasperski & Zielinski [KZ08] investigated robust versions of network optimization problems. From previous work, the question remained open, whether an efficient algorithm achieving a constant performance ratio exists, in case the number of scenarios is unbounded. The authors addressed this question for Min-Max (Regret) versions of classical network optimization problems, namely the SHORTEST PATH, the MINIMUM SPANNING TREE, the MINIMUM ASSIGNMENT, and the MINIMUM $s-t$ CUT. For all the above problems and for both robustness criteria, the authors proved that there is no polynomial algorithm achieving a constant performance ratio unless NP $\subseteq$ DTIME($n^{\text{poly}\log n}$). The last inclusion is widely believed to be untrue.

4.2 Robust single machine scheduling

In this chapter, we will consider the following robust versions of the previously considered scheduling problems.

Problem 4.2.1 (MIN-MAX $1||\sum_j w_j C_j$)

Given: *A set of* jobs $N = \{j_1, j_2, \ldots, j_n\}$ *and a set of k scenarios* $S = \{s_1, \ldots, s_k\}$. *For each job* j_i, *each scenario* s_ℓ *defines a processing time* $p_i^{s_\ell} \in \mathbb{Q}$ *and a weight* $w_i^{s_\ell} \in \mathbb{Q}$.

Find: *An ordering L of the jobs N such that the sum of weighted completion times in the solution's worst-case scenario* $s_{M(L)}$

$$\sum_{i=1}^{n} w_i^{s_{M(L)}} C_i^{s_{M(L)}}$$

is minimized, where $C_i^{s_{M(L)}}$ is the completion time of job j_i when the processing times are defined by scenario $s_{M(L)}$.

We illustrate this definition with an example. Assume an instance with 4 jobs $\{j_1, j_2, j_3, j_4\}$ and two scenarios s_1, s_2. Each scenario defines the processing times and weights of the jobs, as given in Table 4.1.

		j_1	j_2	j_3	j_4
s_1	p :	1	2	3	4
	w :	4	3	2	1
s_2	p :	4	3	2	1
	w :	1	2	3	4

Table 4.1. An example instance of Problem 4.2.1.

First, observe that, if s_1 was the only scenario of this instance (i.e. the problem was given in the classical, non-robust form), the optimal schedule could be computed using Smith's rule (see Theorem 2.1.3). The resulting schedule $L_1 = (j_1, j_2, j_3, j_4)$ evaluates to

$$\text{val}(L_1, s_1) = 4 + 9 + 12 + 10 = 35,$$

which is the optimal value for this scenario. However, the same schedule L_1 has a very high value in scenario s_2, namely

$$\text{val}(L_1, s_2) = 4 + 14 + 27 + 40 = 85,$$

resulting in a high value of L_1 in the robust setting, since

$$\text{val}(L_1) = \max\{\text{val}(L_1, s_1), \text{val}(L_1, s_2)\} = \max\{35, 85\} = 85.$$

The situation is analogous for the schedule $L_2 = (j_4, j_3, j_2, j_1)$ which is optimal for the scenario s_2 but has a high value for scenario s_1. In contrast, an ordering that achieves a "balance" between the two scenarios achieves a much better value. For example, scheduling the jobs according to $J_3 = (j_2, j_4, j_1, j_3)$ evaluates to

$$\text{val}(L_3) = \max\{\text{val}(L_3, s_1), \text{val}(L_3, s_2)\} = \max\{60, 60\} = 60.$$

This is indeed the optimal schedule for this robust problem.

We will also consider the robust version of the problem $1|\text{prec}|\sum_j w_j C_j$, defined as below.

Problem 4.2.2 (MIN-MAX $1|\text{prec}|\sum_j w_j C_j$)

Given: *A set of* jobs $N = \{j_1, j_2, \ldots, j_n\}$*, a set of precedence constraints given by a partial order* $\mathbf{P}(N, P)$ *and a set of* k *scenarios* $S = \{s_1, \ldots, s_k\}$*. For each job* j_i*, each scenario* s_ℓ *defines a processing time* $p_i^{s_\ell} \in \mathbb{Q}$ *and a weight* $w_i^{s_\ell} \in \mathbb{Q}$*.*

Find: *An ordering* L *of the jobs* N *that complies with the precedence constraints such that the sum of weighted completion times in the solution's worst-case scenario* $s_{M(L)}$

$$\sum_{i=1}^{n} w_i^{s_{M(L)}} C_i^{s_{M(L)}}$$

is minimized, where $C_i^{s_{M(L)}}$ *is the completion time of job* j_i *when the processing times are defined by scenario* $s_{M(L)}$*.*

4.3 Bounded number of scenarios

In this section we investigate the problem MIN-MAX $1||\sum_j w_j C_j$ when the number of scenarios in the description of the instance are upper-bounded by some constant. We discuss its complexity in Section 4.3.1 and present an efficient algorithm based on dynamic programming for a special case of this problem in Section 4.3.2.

4.3.1 Complexity of robust scheduling

The computational complexity of the problem MIN-MAX $1||\sum_j w_j C_j$ was already studied by Kuvelis & Yu [KY97], who showed that it is NP-hard even for the case of 2 scenarios. Their proof is by reduction from the (weakly) NP-hard problem 2-PARTITION. Thus, the question about the existence of a fully polynomial time approximation scheme remains open. In the next section we show that, when the numerical parameters of the problem are upper-bounded by some constant, the problem can be solved optimally using dynamic programming.

4.3.2 Efficient algorithm for bounded parameters

In this section we assume that the number of scenarios m as well as the weights and processing times are bounded by some constant. Given an instance I of the robust scheduling problem, let W be the maximum weight and P the maximum processing time occurring in the description of I. We present a polynomial time

algorithm that solves this problem. In fact, we are going to solve the related multi-criteria scheduling problem. This result carries over to our problem by use of the following (more general) Theorem, due to Aissi et. al. [ABV06].

Theorem 4.3.1 (Multicriteria and Robust Approximation) *For any problem* Π *If* MULTI-CRITERIA Π *has a polynomial* α*-approrimation algorithm, then also* MIN-MAX Π *has a polynomial* α*-approximation algorithm.*

In the context of multi-criteria optimization, given two vectors $v, w \in \mathbb{N}^k$, $v \neq w$, $k > 0$, we say that v *dominates* w, if $v_i \leq w_i$ for all $1 \leq i \leq k$. A vector that is not dominated is called *efficient*. Analogously, given a set of vectors S, a subset $S' \subseteq S$ is called an *efficient set* if there is no pair $(v, v'), v \in S, v' \in S'$ such that v dominates v'. The goal in multi-criteria optimization is to find a maximal efficient set of solutions.

For a fixed set of scenarios $S = \{s_1, \ldots, s_m\}$ and a solution L, we will define the *multivalue of* L, denoted

$$\text{mval}(L) = \big(\text{val}(L, s_1), \ldots, \text{val}(L, s_m)\big)$$

as the vector containing the values of L in the different scenarios in an arbitrary but fixed order. Furthermore, we call $\alpha = \big((w_1, p_1), \ldots, (w_m, p_m)\big)$ with $1 \leq w_i \leq W$, $1 \leq p_i \leq W$ a *job profile*, and define the auxiliary functions $p(\alpha) = (p_1, \ldots, p_m)$ and $w(\alpha) = (w_1, \ldots, w_m)$. Note that, since we assumed that P,W and m are all bounded by a constant, there can only be a constant number k of different job profiles. Let $\alpha_1, \ldots, \alpha_k$ be the different job profiles that occur in instance I. We can now identify the instance I by the concatenation of two vectors $(\alpha_1, \ldots, \alpha_k, n_1, \ldots, n_k)$ where n_i is the number of jobs with profile α_i in I. This allows for an easy representation of subinstances of I which will be useful in the following.

We present a dynamic programming approach for solving this version of MIN-MAX $1||\sum_j w_j C_j$ in polynomial time below. Consider a k-dimensional dynamic programming table **DPT** of size $(n_1+1)\times(n_2+1)\times\ldots\times(n_k+1)$. Each cell c of this table represents a subinstance $I(c)$ of I, where the coordinates of c encode the number of jobs of the corresponding profile that are present in $I(c)$. For instance, the cell $(1, 0, 4)$ represents the subinstance of I that contains one job of profile α_1 and four jobs of profile α_3. The total number of jobs in $I(c)$ for $c = (c_1, \ldots, c_k)$ is denoted by $n(c) = \sum_{i=1}^{k} c_i$. Each of these cells will accommodate an efficient set M_c of multivalues of schedules in which only the jobs of the subinstance are considered (note that since the maximum value in any scenario is bounded, there can only be a polynomial number of different efficient vectors). Since the

cell $(n_1, \ldots, n_k)$ represents the whole instance, filling in the last cell of the table would allow us to solve the problem MULTI-CRITERIA $1||\sum_j w_j C_j$, and thus also MIN-MAX $1||\sum_j w_j C_j$. We show how to recursively fill in this table below.

Initialization: We start by filling in the cells representing instances with only one job as follows: for $c_t = 1$ add to M_c the multivalue of the schedule consisting of a single job with profile α_t. This is easily calculated by the pointwise product $p(\alpha_t) \cdot w(\alpha_t)$. We continue filling in the rest of the cells in order of increasing $n(c)$ in the following manner.

Iterative step: Consider the cell $c = (c_1, \ldots, c_k)$ and define

$$T_c = \{(c'_1, \ldots, c'_k) \mid n(c') = n(c) - 1,\ c'_i \geq c_i - 1\}.$$

In other words, T_c contains the cells representing subinstances that result by removing one job from $I(c)$. Note that, since we fill in the table in order of increasing $n(c)$, all cells in T_c have been filled in at this point. For each $c' \in T_c$ with $c_t - c'_t = 1$ and for each schedule L in $M_{c'}$, add to the set M_c the multivalue of the schedule resulting from L by appending a job of profile α_t at the end of the schedule. Given $\mathrm{mval}(L')$, the multivalue of this schedule can easily be computed by:

$$\mathrm{mval}(L) = \mathrm{mval}(L') + w(\alpha_t) \cdot \sum_{i=1}^{k} c_i \cdot p(\alpha_i),$$

where the first multiplication is pointwise and the second a scalar multiplication. Note that only the multivalue of L' is needed in the above calculations, not L' itself. We conclude the computation for cell c by replacing M_c by $Red(M_c)$, which retains only the efficient elements of M_c.

We first prove the correctness of the above algorithm, before analysing it.

Lemma 4.3.2 For every cell c of the table **DPT**, the set M_c is a maximal efficient subset of the set of all multivalues achieved by scheduling the jobs of $I(c)$.

Proof. We need to show that for every cell c of the table **DPT** and every multivalue $\mathrm{mval}(L)$, where L is a schedule of $I(c)$, either

- $\mathrm{val}(L) \in M_c$, or
- $\exists v \in M_c$, such that $v \leq \mathrm{val}(L)$

Assume, towards contradiction, that this is not the case and let c be a cell with minimal $n(c)$ that does not satisfy the above condition. Thus, there is a schedule L of the instance $I(c)$ with $\text{val}(L) \notin M_c$ and for any $v \in M_c$ there is an $\ell \in \{1,\dots,m\}$ with $\text{mval}(L)_\ell < v_\ell$. Clearly, this can only happen for $n(c) \geq 2$. Let α_f be the profile of the job scheduled last in L and let c' be the cell with coordinates $(c_1, c_2, \dots, c_{f-1}, c_f - 1, c_{f+1}, \dots, c_k)$. Furthermore, let L' be the schedule derived from L by omitting the last job. The multivalue of L' is

$$\text{mval}(L) - w(\alpha_f) \cdot \sum_{i=1}^{k} c_i \cdot p(\alpha_i).$$

If there were a $v \in M_c$ such that $\text{val}(v) \leq \text{val}(L')$, then $\text{val}(L)$ would be dominated by $v + w(\alpha_f) \cdot \sum_{i=1}^{k} c_i \cdot p(\alpha_i)$. Thus, for every $v \in M_{c'}$, there is an $\ell \in \{1,\dots,k\}$ such that $v_\ell > \text{val}(L')_\ell$ and thus c' does not satisfy the above property either. Since $n(c') < n(c)$, this contradicts the minimality of c. □

For the analysis of this algorithm, it is easy to see that the initialization of the table, as well as the computations of $\text{val}(L)$ can be done in polynomial time. Furthermore, since $(n^2 \cdot P \cdot W)^2$ is an upper bound on the value of any schedule in any scenario, there can be at most $(n^2 \cdot P \cdot W)^{2m}$ efficient vectors in any stage of the computation. The size of the dynamic programming table is bounded by n^k and for each computation of a cell, at most k cells need to be considered. Moreover, the operator *Red* can be implemented in time $(n^2 \cdot P \cdot W)^{4m}$ by exhaustive comparison. Thus, a single cell can be filled-in in time $k(n^2 \cdot P \cdot W)^{2m} + (n^2 \cdot P \cdot W)^{4m}$, and the whole table in time $n^k \cdot (k \cdot (n^2 \cdot P \cdot W)^{2m} + (n^2 \cdot P \cdot W)^{4m})$. The number of different profiles k is bounded by $(P \cdot W)^m$, which is a constant. Thus our algorithm runs in time $O(n^{8m+W^m P^m})$, i.e. polynomial in n.

4.4 Unbounded number of scenarios

In this section we study the problem Min-Max $1||\sum_j w_j C_j$ for the case that there is no bound on the number of scenarios in the description of the instance. We show that relaxing this condition brings about a big increase in complexity and allows for rather strong inapproximability results. This is in contrast to the classical, non-robust version of this problem, for which obtaining good inapproximability results is considered a prominent open problem.

In Section 4.4.1 we show that Min-Max $1||\sum_j w_j C_j$ cannot be approximated within a better than logarithmic factor, based on a rather standard assumption.

We then restrict the problem to the case that only the weights (or, symmetrically, the processing times) of the jobs are governed by uncertainty. We call this problem "MIN-MAX $1||\sum_j w_j C_j$ with partial uncertainty" and denote it by PARTUNC MIN-MAX $1||\sum_j w_j C_j$. We provide a 2-approximation algorithm for this problem in Section 4.4.3, based on the rounding of a linear program. Surprisingly, this algorithm works even in the presence of precedence constraints, i.e. for the problem PARTUNC MIN-MAX $1|\text{prec}|\sum_j w_j C_j$. Thus, our algorithm matches the best known approximation for the classical, non-robust problem $1|\text{prec}|\sum_j w_j C_j$, and improving it would solve a long-standing open problem in scheduling theory. Furthermore, again contrasting the problem $1|\text{prec}|\sum_j w_j C_j$, we provide rather simple constant inapproximability results for the problem PARTUNC MIN-MAX $1||\sum_j w_j C_j$ in Section 4.4.2.

4.4.1 Inapproximability for the general case

In this section we show that MIN-MAX $1||\sum_j w_j C_j$ cannot be approximated within a factor of $O(\log^{1-\varepsilon} n)$, unless NP has quasi-polynomial time algorithms. The hardness result is obtained by reduction from the version of the LABEL COVER problem defined below. We will need the following definition in order to define LABEL COVER.

Definition 4.4.1 (Total labeling) *Given a bipartite graph $G(V \cup W, E)$, a set of labels $[R]$ and for each edge $(v,w) \in E$ a map $\sigma_{v,w} : [R] \to [R]$, we call a* labeling *of G an assignment of a subset of labels to each of the vertices of G, i.e. a function $\ell : V \cup W \to 2^{[R]}$. We say that a labeling ℓ* satisfies *an edge (v,w) if*

$$\exists a \in \ell(v), \exists b \in \ell(w) : \sigma_{v,w}(a) = b.$$

A total labeling *is a labeling that satisfies all edges.*

Problem 4.4.2 (LABEL COVER $\mathscr{L}(V, W, E, [R], \{\sigma_{v,c}\}_{(v,w)\in E})$)

Given: *A regular bipartite graph $G(V \cup W, E)$ and for each edge $(v,w) \in E$ a map $\sigma_{v,w} : [R] \to [R]$.*

Find: *A total labeling that minimizes* $\max_{x \in V \cup W} |\ell(x)|$.

The value of a Label Cover instance, denoted $\text{val}(\mathscr{L})$, is defined to be the minimum, over all total labelings, of $\max_{x \in V \cup W} |\ell(x)|$. Observe that the variant of the Label Cover problem that is considered assumes that an edge is covered

if, among the chosen labels, there *exists* a satisfying pair of labels. The following hardness result easily follows from the hardness result for the max version by using the "weak duality" relationship between the two versions (see e.g. [AL95]).

Theorem 4.4.3 *There exists a constant $\gamma > 0$ so that for any language L in NP, any input w and any $R > 1$, one can construct a* Label Cover *instance $\mathscr{L}$ with the following properties in time polynomial it the instances size:*

- *The number of vertices in $\mathscr{L}$ is $|w|^{O(\log R)}$ and the number of labels is R.*
- *If $w \in L$ then* $\mathrm{val}(\mathscr{L}) = 1$.
- *If $w \notin L$ then* $\mathrm{val}(\mathscr{L}) > R^{\gamma}$.

Intuitively, the above theorem states that it is NP-hard to distinguish instances of $\mathscr{L}$ that have a covering that uses just one label per vertex from those that require some vertex to be labeled by many labels. Note that the actual resulting hardness depends on the size of the label cover instance produced.

Based on theorem 4.4.3, we will prove the following theorem by presenting a reduction from Label Cover to Min-Max $1||\sum_j w_j C_j$.

Theorem 4.4.4 *There exists a constant $\gamma > 0$ so that for any language L in NP, any input w and any $R > 0$, one can construct a* Min-Max $1||\sum_j w_j C_j$ *instance $\mathscr{S}$ with the following properties:*

- *If $w \in L$ then* $\mathrm{val}(\mathscr{S}) = 1 + o(1)$.
- *If $w \notin L$ then* $\mathrm{val}(\mathscr{S}) = \lceil R^{\gamma} \rceil := g$.

Moreover, the instance $\mathscr{S}$ can be constructed in time $O(|w|^{O(g \log R)} \cdot R^{O(g)})$.

Proof. Given a Label Cover instance $\mathscr{L}(V, W, E, [R], \{\sigma_{v,w}\}_{(v,w)\in E})$, we construct an instance $\mathscr{S}$ of the problem Min-Max $1||\sum_j w_j C_j$. In the next paragraph we give an intuitive description of the reduction, before presenting a more formal proof below.

The following observation leads to the construction of a gadget that is crucial for our reduction. Let $\mathscr{J} := \{J_1, J_2, \ldots, J_k\}$ be a subset of n jobs in a scheduling instance $\mathscr{S}$ and consider the precedence relations $\{J_i < J_{(i+1) \bmod k} \ : \ 1 \le i \le k\}$, forming a cyclical "ordering" of J. Clearly, any schedule of $\mathscr{S}$ induces a linear ordering of the subset J and thus needs to "break" *at least one* of the k precedence relations given above. Now, let us concentrate on an edge $(v, w) \in E$ of $\mathscr{L}$ with its corresponding map $\sigma_{v,w}$. Let R_v, R_w be the possible labels of v and

w respectively, and let $R_{v,w} \subseteq R_v \times R_w$ be the pairs of labels that satisfy $\sigma_{v,w}$, i.e. $R_{v,w} := \{(a,b) \in R_v \times R_w : \sigma_{v,w}(a) = b\}$. Clearly, in any total labeling ℓ of $\mathscr{L}$ there is *at least one* pair $(a,b) \in R_{v,w}$ such that a is a label of v and b is a label of w (we say that (a,b) *covers* (v,w)). Our reduction works by establishing a connection between the two "at least" conditions in the two problems using the gadget described below. The reduction is completed by employing the scenarios in order to synchronize the two objectives with each other.

Gadget construction: Let $(v,w) \in E$ with $n_{v,w} := |R_{v,w}|$. Construct a set $\mathscr{J}^{(v,w)} = \{J_1^{(v,w)}, J_2^{(v,w)}, \ldots, J_{n_{v,w}}^{(v,w)}\}$ of jobs. Each pair of labels in $R_{v,w}$ is assigned to a pair of consecutive jobs using the function $r_{v,w} : R_{v,w} \to \{1, \ldots, n_{v,w}\}$. That is, the i'th pair in $R_{v,w}$ is assigned to the pair $J_i^{(v,w)}, J_{i+1}^{(v,w)}$ (cyclically). Considering the precedence relations $\{J_i^{(v,w)} < J_{(i+1) \bmod n_{v,w}}^{(v,w)} : 1 \leq i \leq k\}$, there must be an index i such that $J_{i+1}^{(v,w)}$ is scheduled before $J_i^{(v,w)}$, as discussed above. See Figure 4.1 for an illustration. In the following we will identify the gadget itself by the set of its jobs $\mathscr{J}^{(v,w)}$.

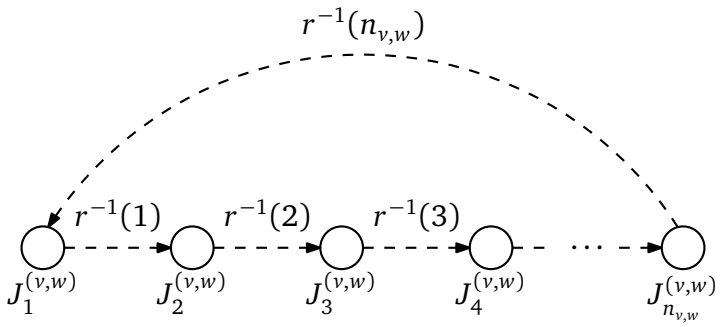


Figure 4.1. An illustration of the gadget $\mathscr{J}^{(v,w)}$ connecting the possible ways of covering of an edge to possible ways of "breaking the cycle" to create a total ordering of the jobs.

Intuitively, a set of scenarios ("counting scenarios", see below) is defined for each edge in such a way that for any possible way of satisfying this edge using at least g labels at some vertex raises the value of a scenario (and thus the maximum) to g. A further set of scenarios ("ordering scenarios") are defined in order to overcome a technical difficulty. A more precise description of the instance $\mathscr{S}$ follows:

Jobs The jobs of instance $\mathscr{S}$ are the union of all jobs in the gadgets, i.e.

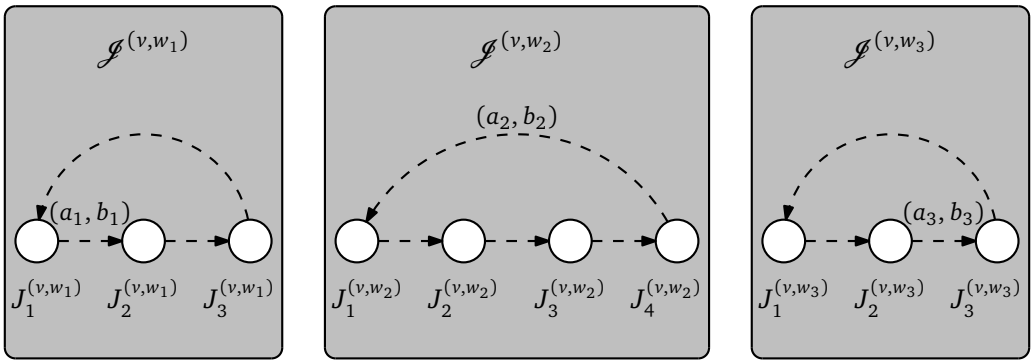
$$\bigcup_{(v,w)\in E} \mathscr{J}^{(v,w)}.$$

Ordering Scenarios This set of scenarios is introduced because it will be convenient to restrict our attention to solutions that schedule the jobs of the gadgets consecutively and in a fixed ordering of the gadgets. This is achieved by making it highly unprofitable to violate this desired property, as follows. Let $m = |E|$ and let $\pi : E \to \{1, \ldots, m\}$ be an arbitrary ordering of the edges. For each $i : 1 \leq i < m$, we have a scenario that sets the weights of the jobs in $\mathscr{J}^{\pi^{-1}(i)}$ to m and the processing time of the jobs in $\bigcup_{j>i} \mathscr{J}^{\pi^{-1}(j)}$ to m. This ensures that any optimal schedule will schedule the jobs in the order

$$\mathscr{J}^{\pi^{-1}(1)} \prec \mathscr{J}^{\pi^{-1}(2)} \prec \cdots \prec \mathscr{J}^{\pi^{-1}(m)}. \tag{4.1}$$

Counting Scenarios This part is the core of our reduction (see also Figure 4.2). We need to define scenarios that get a high value in schedules that correspond to labelings that use many labels in some vertex. In order to achieve this, we have to define a scenario for every possible way that (at least) g labels are assigned to some vertex. For each $v \in V$, each tuple $((v, w_1), \ldots, (v, w_g))$ of g edges incident to v, each subset of g possible labels $(a_1, \ldots, a_g)$ in R_v and each possible choice of $(b_1, \ldots, b_g)$ that satisfies these edges, we define a scenario $\mathscr{S}^{(v,w_1),\ldots(v,w_g)}_{(a_1,b_1),\ldots,(a_g,b_g)}$. This scenario represents the situation that (a_i, b_i) covers (v, w_i) and the number of different labels of v is at least g. This partial label cover solution corresponds to schedules that put $J^{(v,w_i)}_{h+1}$ before $J^{(v,w_i)}_{h}$ in the gadget $\mathscr{J}^{(v,w_i)}$, where $h = r_{v,w_i}(a_i, b_i)$. In order to ensure that scenario $\mathscr{S}^{(v,w_1),\ldots(v,w_g)}_{(a_1,b_1),\ldots,(a_g,b_g)}$ will "count" these g labels of v (i.e. will have value at least g), we define the processing times and weights of the jobs in such a way that the product of the processing time of $J^{(v,w_i)}_{h+1}$ with the weight of $J^{(v,w_i)}_{h}$ equals 1. For instance, we could simply set both values equal to 1. However, there is another technical difficulty that needs to be overcome: due to the ordering scenarios defined above, any job of $\mathscr{J}^{i}$ with non-zero processing time will contribute to the weighted completion time of any job of $\mathscr{J}^{j}, j > i$ the with non-zero weight. This undesirable interaction between the gadgets can be eliminating by setting the weight of $J^{(v,w_i)}_{h}$ to $m^{2\pi(v,w_i)}$ and the weight of $J^{(v,w_i)}_{h+1}$ to $1/m^{2\pi(v,w_i)}$. In this way, the interactions between gadgets become negligible.

The definition of $\mathscr{S}$ is completed by defining scenarios that count the labels for every vertex $w \in W$ in a symmetric way.



$\mathscr{S}^{(v,w_1),(v,w_2),(v,w_3)}_{(a_1,b_1),(a_2,b_2),(a_3,b_3)}$

	p	w
$J_1^{(v,w_1)}$	0	$\frac{1}{m^2}$
$J_2^{(v,w_1)}$	m^2	0
$J_3^{(v,w_1)}$	0	0

	p	w
$J_1^{(v,w_2)}$	m^4	0
$J_2^{(v,w_2)}$	0	0
$J_3^{(v,w_2)}$	0	0
$J_3^{(v,w_2)}$	0	$\frac{1}{m^4}$

	p	w
$J_1^{(v,w_3)}$	0	0
$J_2^{(v,w_3)}$	0	$\frac{1}{m^6}$
$J_3^{(v,w_3)}$	m^6	0

Figure 4.2. An illustration of the definition of a counting scenario, for $g = 3$. For simplicity, we assume that the three considered edges $(v,w_1),(v,w_2)$ and (v,w_3) are put in the beginning by the ordering scenarios.

We conclude the proof by presenting the completeness and soundness analysis.

Completeness Analysis By Theorem 4.4.3, there exists a feasible labeling of $\mathscr{L}$ that assigns one label to each vertex. Let ℓ be such a labeling and consider a schedule σ of I that respects the ordering given by the ordering scenarios (4.1) and such that, for each element $(v,w) \in E$, the jobs in $\mathscr{J}^{(v,w)}$ are scheduled as follows: for $h = 1,\dots,n_{v,w}$, if $h = r_{v,w}(\ell(v),\ell(w))$ then job $J_{h+1}^{(v,w)}$ is scheduled before $J_h^{(v,w)}$, otherwise $J_h^{(v,w)}$ is before $J_{h+1}^{(v,w)}$. This gives a feasible schedule. Moreover, since only one label is assigned to each vertex, it is easy to see that the value of any scenario is at most $1 + o(1)$.

Soundness Analysis Consider a schedule σ of I. Define a labeling ℓ as follows:

$$\ell(v) = \{a : \text{if } J_{h+1}^{(v,w)} \prec J_h^{(v,w)} \text{ for some } h = r_{v,w}(a,b), w \in W \text{ and } b \in R_w\}$$
$$\ell(w) = \{b : \text{if } J_{h+1}^{(v,w)} \prec J_h^{(v,w)} \text{ for some } h = r_{v,w}(a,b), v \in V \text{ and } a \in R_v\}$$

As at least one scenario for each edge will have value 1, ℓ is a feasible labeling for $\mathscr{L}$. Furthermore, by Theorem 4.4.3, there exists a vertex $x \in V \cup W$ so that $|\ell(x)| \geq g$, and this implies that there is a scenario of value g, by construction. Indeed, if $x \in V$ let $\ell(x) = \{a_1, \ldots, a_g\}$ be a set of g labels assigned to x, and let $(b_1, \ldots, b_g)$ and $(w_1, \ldots, w_g)$ be such that $J_{h+1}^{(v,w)} \prec J_h^{(x,w)}$ with $h = r_{x,w_i}(a_i, b_i)$. Then scenario $\mathscr{S}_{(a_1,b_1),\ldots,(a_g,b_g)}^{(x,w_1),\ldots,(x,w_g)}$ has been constructed to have value g according to this schedule. The same holds when $x \in W$.

Size of the instance The total number of scenarios is at most

$$|E| - 1 + 2|E|^g \cdot R^g \cdot R^g$$

and the total number of jobs is at most $|E| \cdot R^2$. As $|E| = |w|^{O(\log R)}$, the total size of the robust scheduling instance is $O(|w|^{O(g \log R)} \cdot R^{O(g)})$. □

By setting $g = O(\log^c n)$ (and $R = O(\log^{O(c)} n)$), where $|w| = n$ and $c \geq 1$ any large constant, we obtain that the input size is equal to $s = n^{O(g \log R)} \cdot R^{O(g)} = n^{O(\log^c n \cdot \log\log n)} \cdot (\log n)^{O(\log^c n)} = n^{O(\log^{c+\delta} n)} = 2^{O(\log^{c+1+\delta} n)}$, for any arbitrarily small $\delta > 0$. It follows that $g = O(\log s)^{\frac{c}{c+1+\delta}} = O(\log s)^{1-\varepsilon}$, for any arbitrarily small $\varepsilon > 0$. We conclude this section by restating the main result.

Theorem 4.4.5 *For every $\varepsilon > 0$, the robust scheduling problem cannot be approximated within ratio $O(\log^{1-\varepsilon} s)$, where s is the input size, unless NP has quasi-polynomial algorithms.*

A note about precedence constraints The above result naturally carries over to the more general problem Min-Max $1|\text{prec}|\sum_j w_j C_j$, i.e. when precedence constraints among the jobs are allowed. However, observe that the two problems are essentially the same when no restriction is imposed on the number of scenarios, in the following sense. Every instance of Min-Max $1|\text{prec}|\sum_j w_j C_j$ can be transformed into an instance of Min-Max $1||\sum_j w_j C_j$ in which the precedence constraints are imposed by additional scenarios, in which the values are chosen in such a way that it becomes highly unprofitable to violate the original precedence constraints. Thus, precedence constraints can be "simulated" by use of additional scenarios. To some extend, this is what we did in the proof of Theorem 4.4.4 when introducing the counting scenarios.

4.4.2 Inapproximability of unweighted case

In this section we consider a more restricted version of MIN-MAX $1|\text{prec}|\sum_j w_j C_j$, namely when only the processing times are affected by uncertainty. We note that this case is symmetric to the one where the processing times are uncertain, while the weights are common to all scenarios. Therefore, we call this problem *partial uncertainty* MIN-MAX $1|\text{prec}|\sum_j w_j C_j$ and denote it by PARTUNC MIN-MAX $1|\text{prec}|\sum_j w_j C_j$. Clearly, the inapproximability presented in Section 4.4.1 does not hold for this case, since both the processing times and weights played a crucial role in our reduction. Nevertheless, we provide very simple constant inapproximability results for this problem, even for the further restricted case that all processing times are equal to one. More precisely, we show that if the number of scenarios is unbounded, the robust scheduling problem is not approximable within 6/5 even for the special case that all processing times are equal to one, and show that this ratio improves to 4/3 assuming the Unique Games Conjecture [Kho02].

We will use the following theorem by Dinur et al. [DGKR03], which establishes the inapproximability of Ek-VERTEX-COVER

Theorem 4.4.6 (Inapproximability of Ek-VERTEX-COVER [DGKR03])
It is NP-hard to decide which of the two categories a given k-uniform hypergraph $G(V,E)$ with $|V| = n$ lies in:

- *k-hypergraphs that have a vertex cover of size $\left(\frac{1}{k-1} + \varepsilon\right) n$*
- *k-hypergraphs whose minimum vertex cover has size at least $(1-\varepsilon)n$*

for an arbitrarily small $\varepsilon > 0$.

We prove the following theorem, which yields the 6/5-inapproximability as Corollary 4.4.8.

Theorem 4.4.7
Given a 3-uniform hypergraph $G(V,E)$, we can construct an instance I of PARTUNC MIN-MAX $1|\text{prec}|\sum_j w_j C_j$ *with n jobs such that:*

- *if G has a vertex cover of size $\left(\frac{1}{2} + \varepsilon\right) n$ then there exists a schedule L of I such that* $\text{val}(L) \leq \left(\frac{5}{2} + \varepsilon\right) n$
- *If any vertex cover of G has size at least $(1-\varepsilon)n$ then any schedule L of I satisfies* $\text{val}(L) > (3-\varepsilon)n$

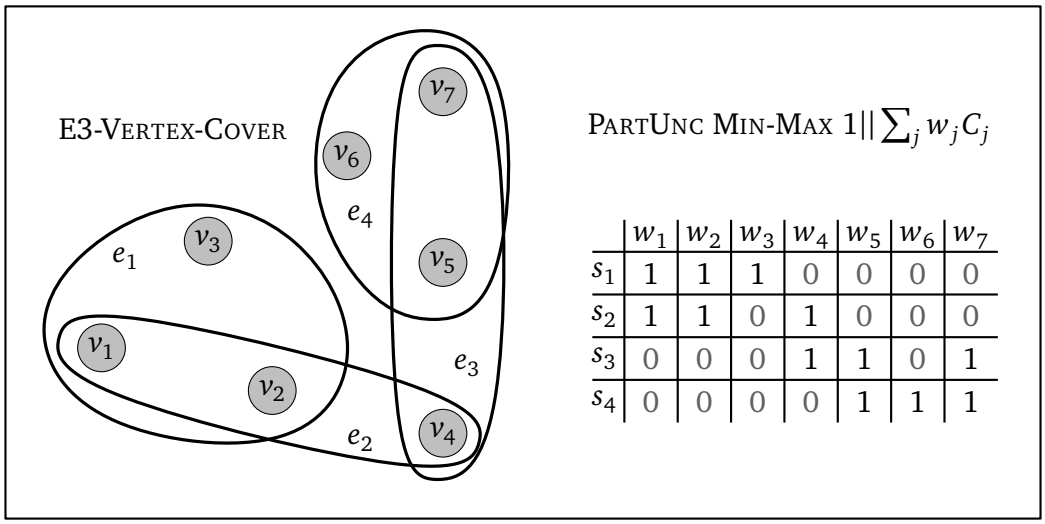
for an arbitrarily small $\varepsilon > 0$.

Proof. We present a reduction from the E3-VERTEX-COVER problem. Given an arbitrary instance of E3-VERTEX-COVER $G(V, E)$ with vertex set V and hyperedge set E, we construct an instance I of PARTUNC MIN-MAX $1||\sum_j w_j C_j$ as follows (see also Figure 4.3).

Jobs For every vertex $v_i \in V$ we create a job $i \in N$ with $p_i = 1$.

Scenarios For every hyperedge $e = \{v_1^e, v_2^e, v_3^e\} \in E$ we create a scenario s_e defined by

$$w_i^{s_e} = \begin{cases} 1 & , \text{ if } v_i \in \{v_1^e, v_2^e, v_3^e\} \\ 0 & , \text{ otherwise.} \end{cases}$$



	w_1	w_2	w_3	w_4	w_5	w_6	w_7
s_1	1	1	1	0	0	0	0
s_2	1	1	0	1	0	0	0
s_3	0	0	0	1	1	0	1
s_4	0	0	0	0	1	1	1

Figure 4.3. An example of the reduction described in the proof of Theorem 4.4.7. Hyperedges are depicted by sets enclosing their incident vertices.

We prove the completeness and soundness of our reduction.

Completeness Assume that there is a vertex cover of size $c \leq \left(\frac{1}{2} + \varepsilon\right) n$, say $C = \{v_1, v_2, \ldots, v_c\}$. Consider any schedule L that schedules the jobs corresponding to vertices in C first. Clearly, since the set C covers all hyperedges, there is no hyperedge all of whose vertices correspond to jobs scheduled after time c. In the scheduling context, this means that there

is no schedule such that the three jobs it assigns unit weight to are scheduled after time c. Thus, the biggest possible value of such a scheduling is attained when the job of C scheduled last is a vertex of a hyperedge whose other two vertices are the last two jobs to be scheduled (see L_1 in Figure 4.4). This gives an upper bound, dependent on c of

$$\mathrm{UB}(c) = c + (n-1) + n < c + 2n \leq \left(\frac{5}{2} + \varepsilon'\right) n$$

for an appropriately chosen $\varepsilon' > 0$. Thus, there exists a schedule L such that $\max_{s \in \mathcal{S}}(L) \leq \left(\frac{5}{2} + \varepsilon\right) n$ for some arbitrarily small $\varepsilon > 0$.

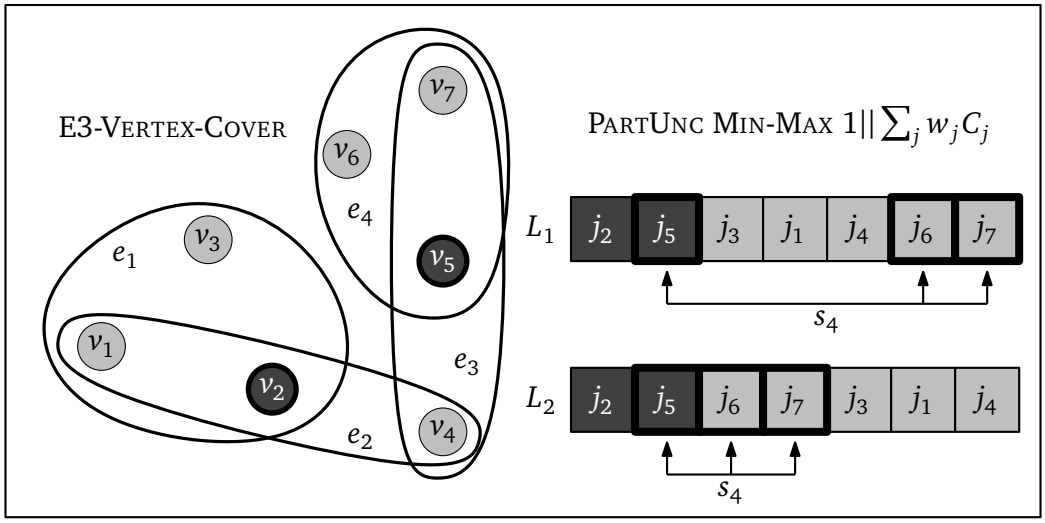


Figure 4.4. Upper (L_1) and lower bounds (L_2) on the value of I constructed by our reduction given the hypergraph on the left. Note that the vertices $C = \{v_2, v_5\}$ form a vertex cover.

Soundness Assume that any vertex cover has size $c > (1-\varepsilon)n$ and consider an optimal schedule L. Define a vertex cover by collecting the vertices corresponding to the jobs scheduled in the beginning, until all hyperedges are covered. Let $C = \{v_1, v_2, \ldots, v_c\}$ be the vertex cover thus constructed. By construction, the last vertex added to this set will be necessary to cover some hyperedge (otherwise the definition of C would have stopped earlier). In other words, there exists a hyperedge whose vertex to be scheduled first is v_c. The corresponding scenario gets its minimum value when the remaining two vertices immediately succeed v_c in the schedule (see

L_2 in Figure 4.4). This gives a lower bound on the value of the optimal schedule L dependent on c of

$$\mathrm{LB}(c) = c + (c+1) + (c+2) > 3c > (3-\varepsilon')n$$

for an appropriately chosen $\varepsilon' > 0$. Thus, any schedule satisfies $\max_{s\in S} > (3-\varepsilon)n$ for some arbitrarily small $\varepsilon > 0$.

□

We now get the inapproximability of 6/5 as a corollary to Theorem 4.4.7, by taking the ratio of the lower to the upper bound. As the unit-time and unweighted robust scheduling problem are symmetric, we can formulate it as follows.

Corollary 4.4.8 [6/5-inapproximability of PARTUNC MIN-MAX $1||\sum_j w_j C_j$] It is NP-hard to approximate the unit-time/unweighted robust scheduling problem within a factor less than 6/5.

We note that the inapproximability of Ek-VERTEX-COVER is strengthened if one is willing to assume the Unique Games Conjecture [Kho02], as shown in [KR08]. A similar reduction as the one described above, this time using 2-uniform hypergraphs (i.e. graphs), leads to a strengthening of our result as well. The reduction is analogous to the one given in the proof of Theorem 4.4.7 and is therefore omitted. Finally, we note that an easy numerical analysis shows that, in both cases, the inapproximability results cannot be improved by changing the uniformity of the hypergraphs in the vertex cover problems considered.

4.4.3 2-approximation for PARTUNC MIN-MAX $1|\mathrm{prec}|\sum_j w_j C_j$

In this section we present a 2-approximation algorithm for the problem PARTUNC MIN-MAX $1|\mathrm{prec}|\sum_j w_j C_j$ based on LP-rounding. We find this result surprising, since PARTUNC MIN-MAX $1|\mathrm{prec}|\sum_j w_j C_j$ contains $1|\mathrm{prec}|\sum_j w_j C_j$ as a single-scenario case. Thus, our result matches the best known result for $1|\mathrm{prec}|\sum_j w_j C_j$ and any improvement would answer a long-standing open question in scheduling theory (see e.g. [SW99]). If in the future the exact approximability of $1|\mathrm{prec}|\sum_j w_j C_j$ is settled at the value 2, our result would state that the approximability of the problem remains unaffected when the weights are replaced by an arbitrary number of scenarios. This strongly contrasts the fact that adding scenarios that also define processing times allows for non-constant inapproximability proofs (see Section 4.4.1).

Consider the following ILP for $1|\text{prec}|\sum_j w_j C_j$ due to Potts [Pot80], already introduced in Section 3.3.2.

$$\begin{aligned}
[\text{P-IP}] \qquad \min \quad & \sum_{j\in N} p_j w_j + \sum_{(i,j)\in N^2} \delta_{ij} p_i w_j && && (4.2a)\\
\text{s.t.} \quad & \delta_{ij} + \delta_{ji} = 1 && \{i,j\} \subseteq N && (4.2b)\\
& \delta_{ij} = 1 && (i,j) \in P && (4.2c)\\
& \delta_{ij} + \delta_{jk} + \delta_{ki} \leq 2 && (i,j,k) \in N^3 && (4.2d)\\
& \delta_{ij} \in \{0,1\} && (i,j) \in N^2 && (4.2e)
\end{aligned}$$

As above, the variables δ_{ij} have the natural meaning that job i is scheduled before job j if and only if $\delta_{ij} = 1$. We could translate this ILP formulation into the robust context by setting the objective as minimizing the value

$$\max_{s\in S} \sum_{j\in N} p_j w_j^s + \sum_{(i,j)\in N^2} \delta_{ij} \cdot p_i w_j^s.$$

This objective, however, is not a linear function. Nevertheless, it can be easily turned into a linear function using an auxiliary variable t that attains the maximum value among all scenarios in an optimal solution, leading to the following formulation.

$$\begin{aligned}
[\text{MM-IP}] \qquad & \min \quad t && && (4.3a)\\
s.t. \quad & \sum_{j\in N} p_j w_j^{s_k} + \sum_{(i,j)\in N^2} \delta_{ij} \cdot p_i w_j^{s_k} \leq t, && 1 \leq k \leq m && (4.3b)\\
& \delta_{ij} + \delta_{ji} = 1 && \{i,j\} \subseteq N && (4.3c)\\
& \delta_{ij} = 1 && (i,j) \in P && (4.3d)\\
& \delta_{ij} + \delta_{jk} + \delta_{ki} \leq 2 && (i,j,k) \in N^3 && (4.3e)\\
& \delta_{ij} \in \{0,1\} && (i,j) \in N^2 && (4.3f)
\end{aligned}$$

The LP relaxation [MM-LP] of the ILP [MM-IP] is obtained by relaxing the constraint $\delta_{ij} \in \{0,1\}$ to $\delta_{ij} \geq 0$. In the next section we first show that this LP has an integrality gap of 2, before giving an LP-rounding based 2-approximation algorithm in Section 4.4.3.

Integrality gap of [MM-LP]

It is easy to see that the resulting LP has an *integrality gap* of 2, as follows. Consider the following family of instances, consisting of n jobs and an equal

number of scenarios. The (scenario-independent) processing times are set to $p_j = 1,\ j \in N$. The weights of the jobs in scenario s_k are defined as follows:

$$w_j^{s_k} = \begin{cases} 1 & \text{, if } j = k \\ 0 & \text{, otherwise} \end{cases},\ j \in N.$$

It is easy to see that setting

$$\delta_{ij} = 1/2, \quad 1 \leq i, j \leq n,\ i \neq j$$

yields a feasible solution. For this solution, all scenarios assume the same objective value

$$p_j + \sum_{i \neq j} \delta_{ij} p_i = 1 + (n-1) \cdot \frac{1}{2} = \frac{n+1}{2}$$

which is trivially also the maximum. This gives an upper bound on the value of the optimal solution of [MM-LP].

On the other hand, for any feasible integral solution, there is a scenario s_k for which the job j is scheduled last. This scenario has value $w_j^k \cdot C_j = n$. Thus the integrality gap of the above presented LP with n scenarios is at least $2n/(n+1)$, which tends to 2 as n tends to infinity. In the next section we provide an LP-based 2-approximation algorithm which shows that our analysis of the integrality gap is tight.

LP-based 2-approximation algorithm

We now provide a 2-approximation algorithm based on rounding the LP-relaxation [MM-LP].

Given an optimal fractional solution $\tilde{\delta}_{i,j}, 1 \leq i, j \leq n$ of the LP, let

$$\tilde{C}_j = p_j + \sum_{i \neq j} \delta_{ij} p_i$$

be the fractional completion time of job j. Assume, without loss of generality, that $\tilde{C}_1 \leq \ldots \leq \tilde{C}_n$. Our proof is based on the following property due to Hall et al. [HSSW97]. We include the proof for the sake of completeness.

Lemma 4.4.9 [Hall et al. [HSSW97]] Given a solution of the above LP, with $\tilde{C}_1 \leq \ldots \leq \tilde{C}_n$ the following inequality holds

$$\tilde{C}_j \geq \frac{1}{2} \sum_{i=1}^{j} p_i$$

Proof. In his dissertation [Sch96a], Schulz proved that the implicitly defined completion times of a feasible solution to [MM-LP] constitute a feasible solution to another linear program formulation using completion times as the decision variables. As a result, feasible solutions of [MM-LP] inherit the property

$$\sum_{j\in S} p_j \tilde{C}_j \geq \frac{1}{2}\left(\sum_{j\in S} p_j^2 + \left(\sum_{j\in S} p_j\right)^2\right), \text{ for every } S \subseteq N.$$

Setting $S = \{1, 2, \ldots, j\}$ implies

$$\sum_{k=1}^{j} p_k \tilde{C}_k \geq \frac{1}{2}\left(\sum_{j\in S} p_j^2 + \left(\sum_{j\in S} p_j\right)^2\right) \geq \frac{1}{2}\left(\sum_{j\in S} p_j\right)^2.$$

Thus

$$\begin{aligned} \tilde{C}_j \sum_{k=1}^{j} p_i &\geq \sum_{k=1}^{j} p_k \tilde{C}_k \\ &\geq \frac{1}{2}\left(\sum_{j\in S} p_j\right)^2 \end{aligned}$$

where the first inequality follows from $\tilde{C}_j \geq C_i$ for $1 \leq i \leq j$. □

This property can be used to derive a simple 2-approximation algorithm: schedule the jobs in non-decreasing order of $\tilde{C}_j$. The integral completion time is

$$C_j = \sum_{i=1}^{j} p_i \leq 2 \cdot \tilde{C}_j.$$

Since every completion time increases by at most a factor of 2, we have a 2-approximate solution.

It is remarkable that the above analysis holds in the presence of precedence constraints, a significant generalization of this problem. For instance, in the single scenario case, the presence of precedence constraints makes the scheduling problem intractable: $1|prec|\sum w_j C_j$ is NP-complete whereas $1||\sum w_j C_j$ is polynomial time solvable. We summarize with the following theorem.

Theorem 4.4.10 *The problem* PARTUNC MIN-MAX $1|\text{prec}|\sum_j w_j C_j$ *has a polynomial time* 2*-approximation algorithm when the processing times or, alternatively, the weights of the jobs do not vary among the scenarios.*

4.5 Conclusions and future research

We have studied different variants of $1||\sum_j w_j C_j$ and $1|\text{prec}|\sum_j w_j C_j$, with bounded and unbounded number of scenarios, in the presence of partial / full uncertainty as well as with bounded weights and unweighted versions. We summarize the results presented in this chapter below:

- Problem Min-Max $1||\sum_j w_j C_j$ (and thus also Min-Max $1|\text{prec}|\sum_j w_j C_j$) with an unbounded number of scenarios cannot be approximated within ratio $O(\log^{1-\varepsilon} n)$ unless NP has quasi-polynomial algorithms.
- Problem Min-Max $1||\sum_j w_j C_j$ with a bounded number of scenarios is solvable in polynomial time when the numerical values of the problem are bounded.
- Problem PartUnc Min-Max $1|\text{prec}|\sum_j w_j C_j$ has a 2-approximation algorithm based on LP-rounding (and this result is hard to improve, as it is a generalization of $1|\text{prec}|\sum_j w_j C_j$).
- Problem PartUnc Min-Max $1||\sum_j w_j C_j$ cannot be approximated within ratio $6/5$ unless P=NP, even when all processing times are equal to 1. Assuming the Unique Games Conjecture, it cannot be approximated within ratio $4/3$.

An interesting question that arises is the approximability of the problem when the number of scenarios is bounded but the processing times and weights are not. It is easy to see that some natural approaches, such as scheduling according to Smith's rule on the average over all scenarios, have an arbitrarily bad approximation ratio in the worst case. It would be interesting to study the approximability of this case more closely.

Moreover, we have proved that the general problem, i.e. when the number of scenarios is not subject to any bounds, cannot be approximated within a better than logarithmic factor. However, we don't know of any positive result for this problem. It would be interesting to investigate whether a matching logarithmic approximation algorithm exists, or stronger inapproximability results can be obtained. As a side-note, while the assumption that NP doesn't have quasi-polynomial algorithms is widely believed to be true, it would be interesting to have a similar result based on a more traditional assumption, such as $\text{P} \neq \text{NP}$.

Finally, while we have shown that the unit processing times case allows for easy inapproximability proofs, there still remains an approximability gap of

$[6/5, 2]$. It is conceivable that a more involved reduction than from hypergraph vertex cover can make better use of the complexity added by the scenarios and close this gap.

Appendix A

Problem Index

Propositional Logic

k-SAT

Given: A boolean formula ϕ in conjunctive normal form with n variables and m clauses, such that each clause contains exactly two literals (i.e. variables or their negations).

Find: A truth assignment for all variables that satisfies all the clauses, i.e. for each clause at least one of its literals is true, or state that no such truth assignment exists.

MAX-k-SAT

Given: A boolean formula ϕ in conjunctive normal form with n variables and m clauses, such that each clause contains exactly k literals (i.e. variables or their negations).

Find: A truth assignment for all variables that maximizes the number of satisfied clauses, i.e. the clauses such that at least one of their literals is true.

Graph Theory

Vertex Cover

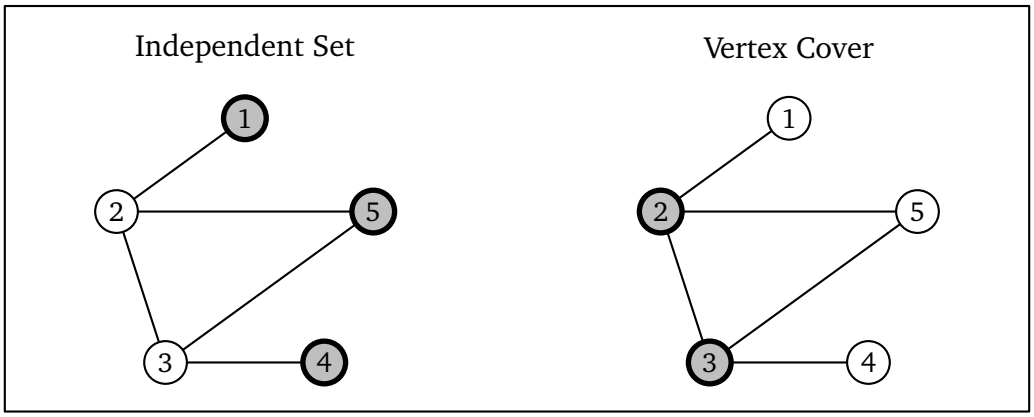
Given: A Graph $G(V, E)$ with vertex set V and edge set E, and a weight function $w : V \to \mathbb{Q}, i \mapsto w_i$.

Find: A *vertex cover* that minimizes the total weight, i.e. a subset $S \subseteq V$ such that for each edge $e = \{v, w\} \in E$, we have $|S \cap \{v, w\}| \geq 1$ and $\sum_{v \in S} w_v$ is minimized.

Independent Set

Given: A Graph $G(V, E)$ with vertex set V and edge set E, and a weight function $w : V \to \mathbb{Q}, i \mapsto w_i$.

Find: An *independent set* that maximizes the total weight, i.e. a subset $S \subseteq V$ such that for each edge $e = \{v, w\} \in E$, we have $|S \cap \{v, w\}| \leq 1$ and $\sum_{v \in S} w_v$ is maximized.



Graph Coloring

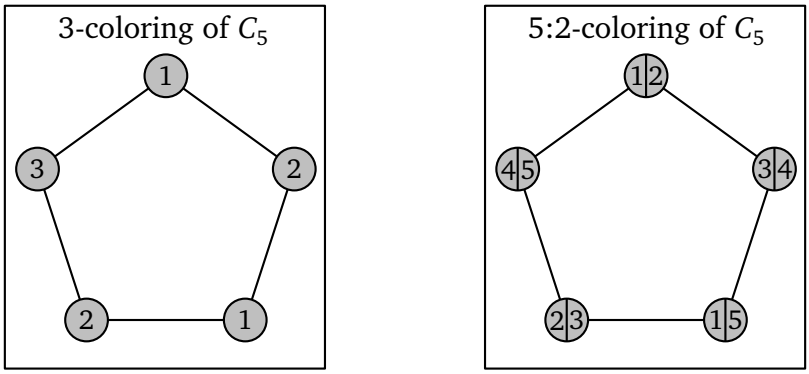
Given: A graph $G(V, E)$ with vertex set V and edge set E.

Find: A k-coloring, i.e. an assignment of colors to the vertices of the graph from a palette of size k such that no two adjacent vertices share the same color and k is minimized.

Fractional Graph Coloring

Given: A graph $G(V, E)$ with vertex set V and edge set E.

Find: An a:b-coloring, i.e. an assignment of a *set* of b colors to each vertex of the graph from a palette of size a such that no two adjacent vertices share a color and a/b is minimized.



Label Cover $\mathscr{L}(V, W, E, [R], \{\sigma_{v,c}\}_{(v,w)\in E})$

Given: A regular bipartite graph $G(V \cup W, E)$ and for each edge $(v, w) \in E$ a map $\sigma_{v,w} : [R] \to [R]$.

Find: A total labeling that minimizes $\max_{x\in V\cup W} |\ell(x)|$.

Unique Games

Given: An undirected, connected graph $G = (V, E)$, a set of colors C and for each edge $\{i, j\}$, $i < j$, a permutation $\pi_{i,j} : C \to C$.

Find: A coloring of the graph that maximizes the number of satisfied edges, i.e. an assignment of colors to vertices $c : V \to C$ such that the number of edges $\{i, j\}$ for which $\pi_{i,j}(c(i)) = c(j)$ holds is maximized.

Scheduling

Load Balancing

Given: A set of m identical *machines*, a set of n *jobs* $N = \{j_1, j_2, \ldots, j_n\}$ and for each job j_i a processing time $p_i \in \mathbb{Q}$.

Find: A partitioning of N into m partitions $N_1, \ldots, N_m$ such that the jobs in N_i are scheduled on machine i in an arbitrary order and without gaps, such that the maximum load among all machines (i.e. the *makespan*)

$$\max_{1 \leq i \leq m} L_i$$

is minimized, where the load of machine i is defined by $L_i := \sum_{j \in N_i} p_j$.

$1||\sum_j w_j C_j$

Given: A set of *jobs* $N = \{j_1, j_2, \ldots, j_n\}$ and for each job j_i a processing time $p_i \in \mathbb{Q}$ and a weight $w_i \in \mathbb{Q}$.

Find: An ordering of the jobs N that minimizes the sum of weighted completion times

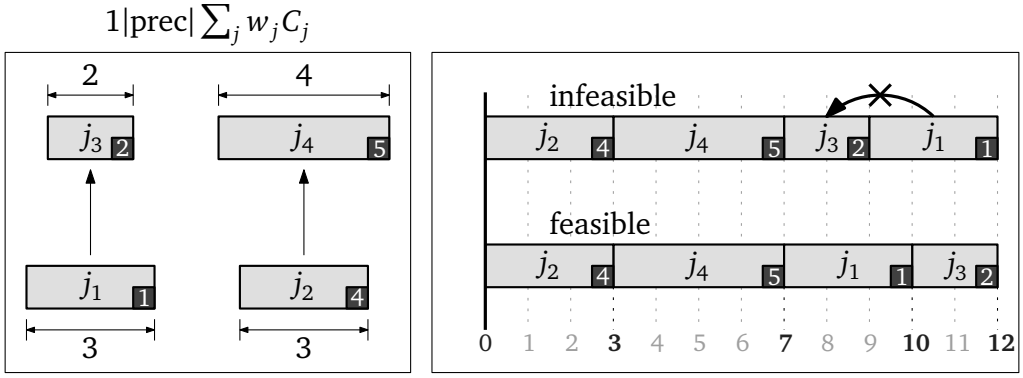
$$\sum_{i=1}^{n} w_i C_i$$

where the completion time C_i of job j_i is the time at which it completes in the schedule.

$1|\text{prec}|\sum_j w_j C_j$

Given: A set $N = \{j_1, j_2, \ldots, j_n\}$ of n jobs, for each job j_i a processing time $p_i \in \mathbb{Q}$ and a weight $w_i \in \mathbb{Q}$, and a partial order $\mathbf{P}(N, P)$ defined on the set of jobs N. If $(i, j) \in P$, i needs to have completed before j can be processed.

Find: A schedule of N without interruptions, i.e. a total ordering L of the jobs on a single machine that respects the precedence constraints and minimizes the weighted sum of completion times $\sum_{i \in N} w_i C_i$.



MIN-MAX $1||\sum_j w_j C_j$

Given: A set of *jobs* $N = \{j_1, j_2, \ldots, j_n\}$ and a set of k scenarios $S = \{s_1, \ldots, s_k\}$. For each job j_i, each scenario s_ℓ defines a processing time $p_i^{s_\ell} \in \mathbb{Q}$ and a weight $w_i^{s_\ell} \in \mathbb{Q}$.

Find: An ordering L of the jobs N such that the sum of weighted completion times in the solution's worst-case scenario $s_{M(L)}$

$$\sum_{i=1}^{n} w_i^{s_{M(L)}} C_i^{s_{M(L)}}$$

is minimized, where $C_i^{s_{M(L)}}$ is the completion time of job j_i when the processing times are defined by scenario $s_{M(L)}$.

MIN-MAX $1|\text{prec}|\sum_j w_j C_j$

Given: A set of *jobs* $N = \{j_1, j_2, \ldots, j_n\}$, a set of precedence constraints given by a partial order $\mathbf{P}(N, P)$ and a set of k scenarios $S = \{s_1, \ldots, s_k\}$. For each job j_i, each scenario s_ℓ defines a processing time $p_i^{s_\ell} \in \mathbb{Q}$ and a weight $w_i^{s_\ell} \in \mathbb{Q}$.

Find: An ordering L of the jobs N that complies with the precedence constraints such that the sum of weighted completion times in the solution's worst-case scenario $s_{M(L)}$

$$\sum_{i=1}^{n} w_i^{s_{M(L)}} C_i^{s_{M(L)}}$$

is minimized, where $C_i^{s_{M(L)}}$ is the completion time of job j_i when the processing times are defined by scenario $s_{M(L)}$.

PARTUNC MIN-MAX $1|\text{prec}|\sum_j w_j C_j$

Given: A set of *jobs* $N = \{j_1, j_2, \ldots, j_n\}$, a set of precedence constraints given by a partial order $\mathbf{P}(N, P)$ and a set of k scenarios $S = \{s_1, \ldots, s_k\}$. For each job j_i, each scenario s_ℓ defines a weight $w_i^{s_\ell} \in \mathbb{Q}$, while its processing time p_i is scenario-independent.

Find: An ordering L of the jobs N that complies with the precedence constraints such that the sum of weighted completion times in the solution's worst-case scenario $s_{M(L)}$

$$\sum_{i=1}^{n} w_i^{s_{M(L)}} C_i$$

is minimized.

Bibliography

[AB09] Sanjeev Arora and Boaz Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, New York, NY, USA, 2009.

[ABV06] Hassene Aissi, Cristina Bazgan, and Daniel Vanderpooten. Approximating min-max (regret) versions of some polynomial problems. In *COCOON*, pages 428–438, 2006.

[AL95] Sanjeev Arora and Carsten Lund. Hardness of approximations. In Dorit S. Hochbaum, editor, *Approximation Algorithms for NP-Hard Problems*. PWS, 1995.

[AM09] Christoph Ambühl and Monaldo Mastrolilli. Single machine precedence constrained scheduling is a vertex cover problem. *Algorithmica*, 53(4):488–503, 2009.

[AMMS07] Christoph Ambühl, Monaldo Mastrolilli, Nikolaus Mutsanas, and Ola Svensson. Scheduling with precedence constraints of low fractional dimension. In Matteo Fischetti and David P. Williamson, editors, *Integer Programming and Combinatorial Optimization, 12th International IPCO Conference, Ithaca, NY, USA, June 25-27, 2007, Proceedings*, volume 4513 of *Lecture Notes in Computer Science*, pages 130–144. Springer, 2007.

[AMMS08] Christoph Ambühl, Monaldo Mastrolilli, Nikolaus Mutsanas, and Ola Svensson. Precedence constraint scheduling and connections to dimension theory of partial orders. *Bulletin of the European Association for Theoretical Computer Science (EATCS)*, 95:45–58, 2008. Appeared online at `http://www.eatcs.org/publications/bulletin.php`.

[AMMS09] Christoph Ambühl, Monaldo Mastrolilli, Nikolaus Mutsanas, and Ola Svensson. Scheduling with precedence constraints with low fractional dimension. *Mathematics of Operations Research*, 2009. Accepted for publication.

[AMS07] Christoph Ambühl, Monaldo Mastrolilli, and Ola Svensson. Inapproximability results for sparsest cut, optimal linear arrangement, and precedence constrained scheduling. In *Proceedings of the 48th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 329–337, 2007.

[BL97] John R. Birge and François Louveaux. *Introduction to stochastic programming*. Springer, 1997.

[BS92] Graham R. Brightwell and Edward R. Scheinerman. Fractional dimension of partial orders. *Order*, 9:139–158, 1992.

[BS02] Dimitris Bertsimas and Melvyn Sim. Robust discrete optimization and network flows. *Programming Series B*, 98:49–71, 2002.

[CH99] Fabián A. Chudak and Dorit S. Hochbaum. A half-integral linear programming relaxation for scheduling precedence-constrained jobs on a single machine. *Operations Research Letters*, 25:199–204, 1999.

[CK98] Pierluigi Crescenzi and Viggo Kann. A compendium of np optimization problems, 1998.

[Cla00] Clay Mathematics Institute. Millenium prize problems – P vs. NP. http://www.claymath.org/millennium/, 2000.

[CM99] Chandra Chekuri and Rajeev Motwani. Precedence constrained scheduling to minimize sum of weighted completion times on a single machine. *Discrete Applied Mathematics*, 98(1-2):29–38, 1999.

[Coo71] Stephen A. Cook. The complexity of theorem-proving procedures. In *STOC*, pages 151–158, 1971.

[CS05] José R. Correa and Andreas S. Schulz. Single machine scheduling with precedence constraints. *Mathematics of Operations Research*, 30(4):1005–1021, 2005.

[DGKR03] Irit Dinur, Venkatesan Guruswami, Subhash Khot, and Oded Regev. A new multilayered pcp and the hardness of hypergraph vertex cover. In *STOC '03: Proceedings of the thirty-fifth annual ACM symposium on Theory of computing*, pages 595–601, New York, NY, USA, 2003. ACM.

[DS02] Irit Dinur and Samuel Safra. The importance of being biased. In *Proceedings of the 34th Annual ACM Symposium on Theory of Computing (STOC)*, pages 33–42, 2002.

[Edm65] Jack Edmonds. Paths, trees, and flowers. *Canadian Journal of Mathematics*, 17:449–467, 1965.

[EEI64] Willard L. Eastman, Shimon Even, and I. M. Isaacs. Bounds for the optimal scheduling of n jobs on m processors. *Management Science*, 11(2):268–279, 1964.

[FJMM07] Uriel Feige, Kamal Jain, Mohammad Mahdian, and Vahab S. Mirrokni. Robust combinatorial optimization with exponential scenarios. In *IPCO*, pages 439–453, 2007.

[FM03] Yaoguang Wang François Margot, Maurice Queyranne. Decompositions, network flows and a precedence constrained single machine scheduling problem. *Operations Research*, 51(6):981–992, 2003.

[FT94] Stefan Felsner and William T. Trotter. On the fractional dimension of partially ordered sets. *DMATH: Discrete Mathematics*, 136:101–117, 1994.

[FT00] Stefan Felsner and William. T. Trotter. Dimension, graph and hypergraph coloring. *Order*, 17(2):167–177, 2000.

[Gas02] William I. Gasarch. The $P =?NP$-poll. *SIGACT News*, 36(2), 2002.

[GJ76] Michael R. Garey and David S. Johnson. The complexity of near-optimal graph coloring. *Journal of the ACM*, 23:43–49, 1976.

[GLLR79] Ronald L. Graham, Eugene L. Lawler, Jan Karel Lenstra, and Alexander H. G. Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of Discrete Mathematic*, 4:287–326, 1979.

[Gra66] Ronald L. Graham. Bounds for certain multiprocessing anomalies. *The Bell System Technical Journal*, 45(9):1563–1581, 1966.

[GW95] Michel X. Goemans and David P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of the ACM*, 42(6):1115–1145, 1995.

[GW00] Michel X. Goemans and David P. Williamson. Two-dimensional Gantt charts and a scheduling algorithm of Lawler. *SIAM J. Discrete Math.*, 13(3):281–294, 2000.

[Hås97] Johan Håstad. Some optimal inapproximability results. In *Proceedings of the 29th Annual ACM Symposium on Theory of Computing (STOC)*, pages 1–10, 1997.

[HJ07] Rajneesh Hegde and Kamal Jain. The hardness of approximating poset dimension. *Electronic Notes in Discrete Mathematics*, 29:435–443, 2007.

[Hoc83] Dorit S. Hochbaum. Efficient bounds for the stable set, vertex cover and set packing problems. *Discrete Applied Mathematics*, 6:243–254, 1983.

[Hoc95] Dorit S. Hochbaum, editor. *Approximation Algorithms for NP-hard Problems*. PWS Publishing Company, Boston, 1995.

[HSSW97] Leslie A. Hall, Andreas S. Schulz, David B. Shmoys, and Joel Wein. Scheduling to minimize average completion time: off-line and on-line approximation algorithms. *Mathematics of Operations Research*, 22:513–544, 1997.

[HSW96] Leslie A. Hall, David B. Shmoys, and Joel Wein. Scheduling to minimize average completion time: Off-line and on-line algorithms. In *Proceedings of the 7th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 142–151, 1996.

[Kar72] Richard M. Karp. Reducibility among combinatorial problems. In Raymond E. Miller and James W. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, 1972.

[Kho02] Subhash Khot. On the power of unique 2-prover 1-round games. In *Proceedings of the 34th annual ACM symposium on Theory of computing (STOC)*, pages 767–775, 2002.

[Kho09] Nikhil Bansal & Subhash Khot. Optimal Long-Code test with one free bit. In *Foundations of Computer Science (FOCS)*, 2009. To appear.

[Knu69] Donald E. Knuth. *The Art of Computer Programming volume 2: Seminumerical algorithms*. Reading, MA: Addison-Wesley, 1969.

[KR08] Subhash Khot and Oded Regev. Vertex cover might be hard to approximate to within $2-\varepsilon$. *Journal of Computer and System Sciences*, 74(3):335–349, 2008.

[KS02] Stavros G. Kolliopoulos and George Steiner. Partially-ordered knapsack and applications to scheduling. In *Proceedings of the 10th Annual European Symposium on Algorithms (ESA)*, pages 612–624, 2002.

[KY97] Panos Kouvelis and Gang Yu. *Robust Discrete Optimization and Its Applications*. Kluwer Academic Publishers, 1997.

[KZ07] Adam Kasperski and Paweł Zieliński. On the existence of an fptas for minmax regret combinatorial optimization problems with interval data. *Operations Research Letters*, 35(4):525–532, 7 2007.

[KZ08] Adam Kasperski and Paweł Zieliński. On the approximability of minmax (regret) network optimization problems. *CoRR*, 2008.

[Law78] Eugene L. Lawler. Sequencing jobs to minimize total weighted completion time subject to precedence constraints. *Annals of Discrete Mathematics*, 2:75–90, 1978.

[Lev73] Leonid A. Levin. Universal search problems. In *Problemy Peredaci Informacii*, volume 9, pages 265–266, 1973. (in Russian).

[LLKS93] Eugene L. Lawler, Jan Karel Lenstra, Alexander H.G. Rinnooy Kan, and David B. Shmoys. Sequencing and scheduling: Algorithms and complexity. *Handbook in Operations Research and Management Science*, 4:445–522, 1993.

[LR78a] Jan Karel Lenstra and Alexander H. G. Rinnooy Kan. Complexity of scheduling under precedence constraints. *Operations Research*, 26:22–35, 1978.

[LR78b] Jan Karel Lenstra and Alexander H. G. Rinnooy Kan. The complexity of scheduling under precedence constraints. *Operations Research*, 26:22–35, 1978.

[MMS08] Monaldo Mastrolilli, Nikolaus Mutsanas, and Ola Svensson. Approximating single machine scheduling with scenarios. In Ashish Goel, Klaus Jansen, Jos'e D. P. Rolim, and Ronitt Rubinfeld, editors, *11th Intl. Workshop on Approximation Algorithms for Combinatorial Optimization Problems - APPROX 2008*, volume 5171 of *Lecture Notes in Computer Science*, pages 153–164. Springer, 2008.

[Möh89] Rolf H. Möhring. Computationally tractable classes of ordered sets. In I. Rival, editor, *Algorithms and Order*, pages 105–193. Kluwer Academic, 1989.

[MS94] Tze-Heng Ma and Jeremy P. Spinrad. On the 2-chain subgraph cover and related problems. *Journal of Algorithms*, 17(2):251–268, 1994.

[NT73] George L. Nemhauser and Leslie E. Trotter Jr. Properties of vertex packing and independence system polyhedra. *Mathematical Programming*, 6:48–61, 1973.

[NT75] George L. Nemhauser and Leslie E. Trotter Jr. Vertex packings: Structural properties and algorithms. *Mathematical Programming*, 8:232–248, 1975.

[Pas97] Vangelis T. Paschos. A survey of approximately optimal solutions to some covering and packing problems. *ACM Computing Surveys*, 29(2):171–209, 1997.

[PC99] Michael Pinedo and Xiuli Chao. *Operations Scheduling with applications in manufacturing and services*, chapter 1. Irwin McGraw-Hill, 1999.

[Pis92] Nicolai N. Pisaruk. The boundaries of submodular functions. *Computational Mathematics and Mathematical Physics*, 32(12):1769–1783, 1992.

[Pis03] Nicolai N. Pisaruk. A fully combinatorial 2-approximation algorithm for precedence-constrained scheduling a single machine to minimize average weighted completion time. *Discrete Applied Mathematics*, 131(3):655–663, 2003.

[Pot80] Chris N. Potts. An algorithm for the single machine sequencing problem with precedence constraints. *Mathematical Programming Study*, 13:78–87, 1980.

[Pra07] Martin Prado. Amazing baseball bat stands upright vertical. `http://www.youtube.com/watch?v=k6LNnKonTOw`, 2007.

[PY79] Christos H. Papadimitriou and Mihalis Yannakakis. Scheduling interval-ordered tasks. *SIAM Journal on Computing*, 8:405–409, 1979.

[PY91] Christos H. Papadimitriou and Mihalis Yannakakis. Optimization, approximation, and complexity classes. *Journal of Computer and System Sciences*, 43:425–440, 1991.

[Rab78] Issie Rabinovitch. The dimension of semiorders. *J. Comb. Theory, Ser. A*, 25:50–61, 1978.

[Sch96a] Andreas S. Schulz. *Polytopes and Scheduling*. PhD thesis, Department of Mathematics, Technische Universität Berlin, February 1996.

[Sch96b] Andreas S. Schulz. Scheduling to minimize total weighted completion time: performance guarantees of LP-based heuristics and lower bounds. In *Proceedings of the Fifth Conference on Integer Programming and Combinatorial Optimization (IPCO)*, volume 5, pages 301–315, 1996.

[SG76] Sartaj Sahni and Teofilo F. Gonzalez. *P*-complete approximation problems. *Journal of the ACM*, 22:115–124, 1976.

[Sid75] Jeffrey B. Sidney. Decomposition algorithms for single-machine sequencing with precedence relations and deferral costs. *Operations Research*, 23:283–298, 1975.

[Smi56] Wayne E. Smith. Various optimizers for single-stage production. *Naval Research Logistics Quarterly*, 3:59–66, 1956.

[Spe71] Joel Spencer. On minimum scrambling sets of simple orders. *Acta Mathematica*, 22:349–353, 1971.

[SU97] Edward R. Scheinerman and Daniel H. Ullman. *Fractional Graph Theory*. John Wiley and Sons Inc., 1997.

[SU08] Andreas S. Schulz and Nelson A. Uhan. Near-optimal solutions and integrality gaps for almost all instances of single-machine precedence-constrained scheduling. Preprint, 2008.

[Sve08] Ola N. A. Svensson. *Approximability of Some Classical Graph and Scheduling Problems*. PhD thesis, Università della Svizzera Italiana, University of Lugano, 2008.

[SW99] Petra Schuurman and Gerhard J. Woeginger. Polynomial time approximation algorithms for machine scheduling: ten open problems. *Journal of Scheduling*, 2(5):203–213, 1999.

[Tro92] William T. Trotter. *Combinatorics and Partially Ordered Sets: Dimension Theory*. Johns Hopkins Series in the Mathematical Sciences. The Johns Hopkins University Press, 1992.

[Tur36] Alan M. Turing. On computable numbers, with an application to the entscheidungsproblem. *Proceedings of the London Mathematical Society*, 2(42):230–265, 1936.

[Uha08] Nelson A. Uhan. *Algorithmic and Game-Theoretic Perspectives on Scheduling*. PhD thesis, Massachusetts Institute of Technology (MIT), 2008.

[Vaz01] Vijay V. Vazirani. *Approximation Algorithms*. Springer, 2001.

[Wei] Eric W. Weisstein. Algorithm. `http://mathworld.wolfram.com/Algorithm.html`. From MathWorld – A Wolfram Web Resource.

[Woe03] Gerhard J. Woeginger. On the approximability of average completion time scheduling under precedence constraints. *Discrete Applied Mathematics*, 131(1):237–252, 2003.

[Yan82] Mihalis Yannakakis. On the complexity of partial order dimension problem. *SIAM Journal on Algebraic and Discrete Methods*, 22(3):351–358, 1982.

Index

This thesis contains a false statement.