Algorithms for a bounded-width order acceptance and scheduling problem with sequence-dependent setup time using decision diagrams

Robert Baart

Algorithms for a bounded-width order acceptance and scheduling problem with sequence-dependent setup time using decision diagrams

THESIS

submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE

in

COMPUTER SCIENCE

by

Robert Baart
born in Schiedam, the Netherlands

TU Delft

Algorithmics Research Group
Department of Software Technology
Faculty EEMCS, Delft University of Technology
Delft, the Netherlands
www.ewi.tudelft.nl

Algorithms for a bounded-width order acceptance and scheduling problem with sequence-dependent setup time using decision diagrams

Author: Robert Baart
Student id: 4108272
Email: robert.baart@gmail.com

Abstract

Abstract

Order acceptance and scheduling problems (OAS) is a category of problems where the decision of which jobs to schedule and how to schedule them is intertwined. Decision diagrams have in recent years become a popular method of finding bounds for scheduling problems. However, their use in problems where orders can also be rejected has not yet been explored. We investigate the application of decision diagrams to OAS problems by studying a specific problem in detail. The OAS problem considered here is a single machine deterministic problem with sequence-dependent setup time (OAS-SMS) with a further width restriction such that the number of jobs available to be scheduled at any one time is bound by a constant.

We show this problem to be weakly NP-hard and develop an exact decision diagram formulation for it as well as a restricted decision diagram formulation which we show to be a fully polynomial time approximation scheme (FPTAS) of this problem for a constant width. To make decision diagrams work with order acceptance we generalise the decision diagrams to not require explicit layers associated with the decision variables. We also show that this definition of width used here generalises the Balas-Simonetti neighbourhood in travelling salesman problems and that the exact algorithm solves it in the same worst case time complexity. We evaluate the exact and approximation algorithms on an existing OAS-SMS benchmark set and show it exceeds the state of the art on instances of bounded width.

Thesis Committee:

Chair and supervisor:	Dr. M.M. de Weerdt, Faculty EEMCS, TU Delft
Committee Member:	Prof.dr.ir. D.H.J. Epema, Faculty EEMCS, TU Delft
Committee Member:	Dr. N Yorke-Smith, Faculty EEMCS, TU Delft

Preface

Before you lies the thesis that forms the capstone of my master Computer Science at Delft University of Technology. When I started this master after having finished a bachelor in applied physics, it was algorithm design that had drawn me here. The clever tricks and subtle changes of perspective when considering a problem that allow it to be solved much more quickly delighted me and still do, and I enjoyed puzzling over my own problem these past eight months. I could not have done so without the help of many people, and I would like to thank the following people in particular.

First and foremost I would like to thank dr. Mathijs de Weerdt for suggesting this subject, for the many discussions and, nearer the end of my work, for telling me to actually write it all down. My gratitude also goes to prof. Dick Epema and dr. Neil Yorke-Smith for being part of my thesis committee.

Next I would like to thank Lei He for a helpful introduction to the order acceptance and scheduling problem, and thank both him and Pim van den Bogaerdt for great conversations of which I should have had more. Thanks also to my friends in whose company I studied, you made the many hours spent on this thesis enjoyable.

My parents and brother I would like to thank for providing a day in the week where I could clear my head and think of something else. For every other day, for all the messages of encouragement and for keeping up with my I would like to thank my boyfriend Jens de Waard, without whom I would not know how to do life well, let alone this thesis.

To all the friends and family who have not seen me much for the past few months, who mailed me without reply, who did not see me at their events... Well, I'm back.

Robert Baart
Delft, the Netherlands
November 23, 2018

Contents

Chapter 1

Introduction

In the order acceptance and scheduling problem (OAS) a set of jobs is available from which a subset must be selected and scheduled, and the decision of which jobs to accept and how to schedule them is intertwined. This models many real life situations where the processing capacity is limited and neither choosing a subset of jobs to accept nor scheduling the chosen jobs is a trivial task. In general the acceptance and scheduling parts of these problems cannot be solved separately to an optimal combined solution, as the feasability or value of a chosen subset involves finding their optimal schedule and the optimal schedule may change completely with the addition or removal of a single job.

There are many variants of the OAS problem. As it contains a scheduling component, much of the scheduling terminology translates directly to these problems. They can be categorised by the same major scheduling categories of the number of machines, the constraints and objective function. However, common objective functions in scheduling literature such as makespan, lateness and tardiness do not translate directly since not all jobs need to be scheduled. In a taxonomy of OAS problems [25] it can be seen that the approach taken by many is to define some cost or profit function that rewards job acceptance and punishes tardiness, which can be done in a multitude of ways.

With some exceptions [23, 18], problems in the family of OAS problems are often NP-hard, meaning they are too difficult to be solved exactly in a timespan bounded by a function polynomial in the size of the problem. There are three main approaches to still achieve progress on such problems. One can look for exact algorithms that are still exponential in the worst case but can solve more instances or slightly larger problem sizes than earlier algorithms. Alternatively, one can create heuristic algorithms that find decent solutions in many cases though with no guarantee of doing so. The third approach is to seek to shrink the difficult class of problems by finding a subclass of problems which is more easily solved due to some property it has and develop an algorithm that exploits this property. This is useful in its own right if that property is common in real-world applications, but can also be used to solve subproblems in a branching or local search approach to the general difficult class of problems. This last approach is quite common in other scheduling and permutation finding problems [7, 11], but not so in recent OAS literature [25].

A motivating problem for this thesis is a satellite observation scheduling problem. A satellite equiped with a camera is in orbit over the earth and has a set of jobs to perform in

the form of images to take. More advanced *agile* earth observing satellites can move the direction their camera is pointing side to side so the places they take pictures of do not have to lie on a line, and some can move the camera along all axes, allowing them to look ahead and back as well.

The ability to move along more than one axes results in a availability window for each image to be taken. A limit on the movement speed of the camera as well as the required time to take a picture sets further time constraints, which can make it impossible to find a feasible schedule for all images. Together this makes satellite observation scheduling an OAS problem, but with the interesting property that jobs lie along a predefined overall path, in this case the orbit. Jobs that lie far apart in this overall path will not directly compete for a specific execution time, but can influence the possible further paths through their completion time and associated camera position [17].

There are other problems that have a similar overall path. Once such an overall route or destination is chosen, the jobs that lie along it could also lie sufficiently far apart that they only influence each other through the setup or travel time between jobs and through delays. This can be an inherent property of the problem as it is with satellite scheduling and might be for fixed delivery routes or transport along a river or railway. It can also be a subproblem to a more general scheduling problem, such as a prize collecting travelling salesman problem where in a branching approach the overall route is already chosen but the ordering at a local level not fully determined.

In this thesis we focus on an abstraction of the satellite observation scheduling problem as a deterministic single machine OAS problem which is known to be *strongly NP-hard* [22]. The concept of an overall path is formalised as a *width* restriction on the number of overlapping availability windows, which creates a subclass of the OAS problem. We analyse the complexity of this subclass and design exact and approximation algorithms for it using decision diagrams.

Decision diagrams have recently become popular in optimisation [5], as they form an intuitive extension of dynamic programming and particularly the state transition graphs thereof. They have also shown to be useful in single machine scheduling problems and similar sequencing problems [9, 10], though not yet in the order acceptance and scheduling family of problems where orders may be rejected. We suspect that this specific subclass with a bounded width is particularly suitable to being solved using decision diagrams, This is because a natural representation of the problem as a directed graph, with vertices representing jobs and edges possible orderings, closely resembles a decision diagram.

The abstraction of the satellite observation scheduling problem models the images to take as jobs with a release time and a hard deadline and processing time for the time it takes to take a picture. It also models the required movement time of the camera between images as a setup time for each direct ordering of two jobs. Each image has a certain value, a schedule is optimal if it maximises the summed value and is feasible. The number of jobs available at any one time is limited by the width, which enforces a broad ordering.

The question we seek to answer is the following. **Can decision diagrams aid in finding good upper and lower bounds on bounded-width order acceptance and scheduling problems?** To answer this question we split the question in the following three parts, of which the first is necessary preliminary work. First we examine the complexity of the cho-

sen bounded-width problem to confirm that it remains NP-hard. Secondly we formulate an exact decision diagram approach to solving this OAS problem and then relaxed and restricted versions of it that can run in polynomial time. These are then evaluated by comparing them to existing algorithms for the more general OAS problem without bounded width. A crucial subquestion of the second part is whether decision diagrams can be efficiently adapted to problems where orders may be rejected.

The rest of this thesis consists of the following chapters. Chapter 2 starts with a background of the particular OAS problem we examine, preceding research in this topic area and briefly summarises decision diagrams so their usage is clear in the later chapters. Chapter 3 contains the theoretical development of this work with a complexity proof of this special case of OAS-SMS and the development of an exact exponential algorithm and a FPTAS. In chapter 4 the experimental setup and the datasets used to test the algorithms are presented and examined and in chapter 5 the experimental results are shown and analysed. The last chapter concludes with a summary of what was achieved and suggests further research that could build on these results.

Chapter 2

Background

In this chapter we will give a formal definition of the specific order acceptance and scheduling (OAS) problem considered in this thesis, including the width restriction that we use in this thesis. An overview is given of the preceeding work done on this particular problem as well as on some similar problems in other areas. We then give a short background on decision diagrams, their definition, usage and variations, as these will be used in the development of algorithms.

2.1 The order acceptance and scheduling problem considered

The chosen order acceptance and scheduling problem chosen to evaluate the use of decision diagrams on is single machine OAS with sequence-dependent setup times (OAS-SMS) [22]. This is a deterministic, offline problem which has the release times and deadlines that allow for a width restriction. It also has sequence-dependent setup times which gives the problem a travelling salesman component.

The setup times, release times and deadlines also make it a good model for the satellite observation scheduling problem that inspired the width restriction, as jobs are only available when the satellite is near it in its orbit, and the transition time between images depends on the angle between the two.

The exact definition is as follows. A set of n jobs is given $J = \{1 \dots n\}$. Each job i has a release date r_i, a due date d_i and deadline $\overline{d}_i$, processing time p_i, value e_i, penalty weight w_i and for each pair of different jobs i and j there is a setup time s_{ij}. There is also a setup time for jobs executed first in a solution s_{0i}, but not after completion of the last job. The penalty weight w_i is chosen such that the value becomes zero at the deadline, so $w_i = e_i/(\overline{d}_i - d_i)$. All values are positive, and all but the penalty weights are integers. The goal is to find a schedule S, consisting of a subset of J and for each job i in that subset an associated start time t_i, which is feasible and optimal. Feasible here means that for each job $i \in S$, $r_i \leq t_i \leq \overline{d}_i - p_i$ and for each pair of consecutive jobs $i, j \in S$ that $\min(t_i + p_i, r_j) + s_{ij} \leq t_j$. We say that the time window $\left[r_j, \overline{d}_j - p_j\right]$ is the *availability window* of job j. An optimal solution is defined here as one that maximises revenue, or $\sum_{i \in S} e_i - w_i \max(0, t_i + p_i - d_i)$, the sum of value of the scheduled jobs minus each of their tardiness penalties if incurred

Table 2.1: Example OAS-SMS problem consisting of 4 jobs. The setup times are not shown in the table, they are $s_{ij} = 1$ for all i, j except $s_{04} = s_{41} = s_{43} = 2$.

i	r_i	p_i	d_i	$\overline{d}_i$	e_i
1	0	3	7	8	2
2	2	2	9	10	3
3	3	3	8	10	4
4	6	2	13	14	2

[22]. To facilitate reasoning about this problem, a source and a sink job of zero revenue and processing time are added with release times before and after all other jobs respectively. This formulation is ambiguous about the revenue for a job completed at a due date that equals the deadline, earlier work got around this by never having due dates equal to the deadline. We will assume that in such cases the full value is awarded.

A solution to this problem is given by a schedule S consisting of an ordered subset of jobs and associated with each job in this ordered subset a start time. The value of a solution is the summed revenue of all jobs in the subset started at those times, as shown in equation 2.1.

$$\sum_{i \in S} e_i - w_i \max(0, t_i + p_i - d_i) \tag{2.1}$$

A solution, aside from consisting of a subset of jobs ordered by ascending start time, must obey two main constraints. Constraint 2.2 that jobs must be started after their release time and complete on or before their deadline. Constraint 2.3 is on setup times between consecutive jobs. Because they only start after the release time, the release time of the previous job is relevant to this constraint.

$$r_i \leq t_i \leq \overline{d}_i - p_i \quad \forall i \in S \tag{2.2}$$

$$\min(t_i + p_i, r_j) + s_{ij} \leq t_j \quad \forall i, j \in S \text{ if } i \to j \tag{2.3}$$

This is a strongly NP-hard problem, so an optimal solution that scales well is unlikely. Instead we focus on a special case of the OAS-SMS problem where the number of jobs that could start at any time point is limited by a constant width. This allows us to focus on a subset of problems potentially ill served by algorithms that need to take into account very wide instances as well, like the satellite scheduling problem and may lead to a good local search approach.

The exact restriction is as follows. We define width for a time point t as the number of jobs in J whose availability window includes the time point t. The width restriction imposed is there can be no time point for which the width exceeds w, where w is a constant independent of the number of jobs n, and in itself also a property of the problem.

An example OAS-SMS problem consisting of 4 jobs is shown in table 2.1. This problem has a width of 3, as there is no time jobs 1 and 4 both could start. The optimal sequence is

$1 \rightarrow 3 \rightarrow 4$ which completes at time $t = 11$ and has a revenue of 8. However, if this problem is seen as the first 4 jobs of a larger problem, then it has multiple sequences that may later lead to an optimal solution, as some with lower revenue do complete earlier potentially allowing more lucrative subsequent jobs.

This variant of the order acceptance and scheduling problem is first introduced by Og et al. [22] and they showed it to be strongly NP-hard. They formulated a mixed integer linear program, which was used for generating upper bounds on the optimal solutions and can solve instances optimally for up to 15 jobs, and they introduce two heuristic algorithms to find feasible solutions that are close to the upper bound. In a review paper [25] a taxonomy of different OAS variants is given as well as a review of research up to that point on the topic. In it, OAS-SMS is included in the category of deterministic single machine OAS problems, with as its distinguishing features the setup times and presence of release times and deadlines.

Cesaret et al. [8] improve on the heuristic algorithms with a tabu search approach that solves instances much faster and finds solutions that are closer to the upper bounds. They generated a new benchmark set using the same procedure as [22] used for theirs though with diferent parameters. They made it publicly available for subsequent use. More heuristic algorithms were proposed by Nguyen et al. [21] and Nguyen [20], both based on a genetic algorithm with one using a dispatching rule and local search and the other a hyper heuristic over the genetic algorithm. Their results were competitive with those of Cesaret et al..

In a recent paper [24] new exact and heuristic algorithms are presented which give tighter bounds and competitive solutions respectively. Of the problems in the benchmark set [8] they find the optimal solutions for most instances except those with $n = 100$ jobs. These remain too hard to solve within one hour for all exact algorithms proposed so far. The best bounds on them have a gap between the lower and upper bounds in the vicinity of 10%. The CPU time of their exact algorithm performs better on larger values of τ, one of the parameters of the dataset they use. A larger τ is associated with a smaller width which could affect runtime, but the difference in CPU time is not very large.

The ability to find a schedule which is optimal, or at least closes the current gap of 10% further between the lower and upper bound, would still represent a sizeable increase of revenue in proportion to the current 90% efficient lower bounds in these environments. The current upper limit of $n = 50$ jobs which can be solved to optimality is also not yet so high that receiving more orders is uncommon. Aside from solving larger instances, solving the smaller ones faster would open up more applications. As the problem is strongly NP-hard, there is also value in identifying subclasses of the problem that can be solved efficiently but are not currently by existing algorithms.

2.1.1 Similar problems

The OAS-SMS is closely related to a variant of the Travelling Salesman Problem (TSP), specifically the Prize Collecting TSP with Time Windows. The one major difference between the two is that the setup time of OAS-SMS only starts after the release time of a job, whereas the travel time between cities in TSP does not. This closely related problem is of interest here because the exploration of special cases, a common area of research for

TSP problems, is not found as readily in OAS research which focuses more on heuristic algorithms [25].

In particular Balas and Simonetti [4] use a similar width restriction on closely related TSP problems, though the width definition we use here is defined differently and order acceptance is not part of their consideration. In another paper [26] a direct analogue of the width restriction used here is considered for travelling salesman and travelling repairman problems with time windows. Others use some structure in the the distance between cities, which in the OAS problem translate to setup times [1]. These provide some inspiration for how structures in a more general problem can be exploited to solve the problem more quickly than would be possible otherwise.

2.2 A background on decision diagrams

Decision diagrams are compact representations of decision trees, where vertices that for all possible subsequent decisions lead to the same outcome are merged, forming a directed acyclic graph. These were originally used to compactly represent logic circuits [2], but have recently started to be used for optimisation. Advantages of decision diagrams over the similar approach of dynamic programming is that their structure does not require enumeration and allows for solution space relaxation or restriction on the fly.

Decision diagrams can be used as part of a branch and bound approach to discrete optimisation problems [6] by providing upper or lower bounds for which relaxed, or sometimes restricted decision diagrams, are used.

2.2.1 Definition and notation

Let P be a problem instance and $X = \{x_1, \ldots, x_n\}$ an ordering of its decision variables, which should have a discrete and finite domain. A decision diagram for (P, X) is a directed acyclic graph $G = (V, E)$ that encodes all feasible solutions. The vertices V are all associated one to one with a decision variable, all vertices associated with the same decision variable x_i are said to be in the same layer L_i. There is one vertex r in layer L_0 with no associated decision variable, this *source* vertex represents the state of not having taken any decisions yet.

For each vertex v there is an associated *state* $S(v)$ which represents what combinations of subsequent decisions are still available. There are no two vertices with the same state. This means the last layer L_n has just one vertex, the *sink*. There are only directed edges from a node in one layer to a node in the next layer. Each edge (v, w) from layer L_{i-1} to L_i has an associated label which is a valid assignment to decision variable x_i given state $S(v)$ and results in available subsequent decisions represented by state $S(w)$.

Any path of edges from the source vertex therefore represents a feasible assignment to the decision variables of the layers in that path, a *partial solution* consisting of the cost or reward of all decisions so far and a state representing the feasible subsequent decisions. A source-sink path of edges is a feasible solution to P, and every feasible solution is represented by some source-sink path.

The use of decision diagrams for problems that allow order rejection is not yet explored a lot. The natural decision associated with each layer for sequencing problems is which job

to sequence next, but when jobs can be rejected this can lead to identical states in different layers [19].

2.2.2 Relaxed and restricted decision diagrams

The decision diagram defined above is an exact decision diagram, by which is meant that all solutions represented in the decision diagram are feasible, and that all feasible solutions are represented in the decision diagram. In many applications the number of distinct states and by extension the number of vertices needed to represent a problem may grow prohibitively fast for increasing problem size.

There are two approaches to remedy this, which both loose some guarantee the exact decision diagrams has. *Relaxed* decision diagrams loosen the requirement that all solutions in it be feasible, thereby allowing the merging of vertices with different states as long as the state of the combined vertex encompasses at least all the subsequent choices the two separate vertices did. In contrast, *restricted* decision diagrams loosen the requirement that all feasible solutions be represented, so vertices can be merged so long as the combined vertex state does not allow for any infeasible solutions.

A relaxed decision diagram therefore represents a superset of all solutions and its best solution gives an optimistic bound on the optimal solution, whereas a restricted decision diagram represents a subset of all solutions and its best solution is a pessimistic bound on the optimal solution.

Chapter 3

Theoretical development

This chapter contains the theoretical development of a relaxed and restricted decision diagram formulation for the bounded-width OAS-SMS problem, and the required preliminary work to get there. The chapter begins with an analysis of the complexity of the chosen OAS problem and then proves various properties required the correctness of an exact decision diagram formulation. An algorithm is then presented for finding the optimal revenue via the top down compilation of such an exact decision diagram. This is then extended through a vertex merge rule to the construction of a relaxed and a restricted decision diagram where the restricted decision diagram is a FPTAS of the bounded-width OAS-SMS problem.

The main challenges in the development of these decision diagrams lie in the adaptation of decision diagram theory to the concept of order acceptance and rejection. Decision diagrams in literature are partitioned into layers [5], but in the case of order acceptance and scheduling this poses a problem. Choosing for each job whether it should be scheduled is not a sufficient set of decision variables as forming a schedule from the resulting subset scheduling is non-trivial. Deciding on what job to schedule next can result, in a traditional decision diagram, in vertices with the same state being in different layers. This is because not all jobs need to be scheduled, so nothing prevents schedules containing a different number of jobs to have an identical state describing possible future scheduling. After establishing the problem complexity, solving this challange to the use of decision diagrams for OAS problems will be the main focus.

3.1 Problem complexity

The OAS-SMS problem is strongly NP-hard [22]. The problem considered here with the addition of a width restriction is a subclass of the OAS-SMS problem. It therefore cannot be harder than the OAS-SMS problem, but it could belong to any easier complexity class. To draw meaningful conclusions about algorithms developed for this new problem, its complexity should be explored, as the runtime bound of an algorithm can only be evaluated well in the context of the problem's complexity.

3.1.1 Behaviour in edge cases

To explore the complexity of the problem we first discuss the behaviour in extreme cases. We also examine some similar problems that only differ slightly. This can assist in understanding the problem, even if the small differences render reusing solution strategies incompatible.

Ignoring width for the moment, a problem where $r_i = s_{ij} = 0$ and $\overline{d}_i = d_i = d$ is equivalent to the knapsack problem, where time fulfills the role of weight in the knapsack formulation. This illustrates one of the challenges that makes this problem NP-hard to solve optimally, an infinitesimally small amount of more time remaining can be the difference between a lucrative job being schedulable or not. Conversely, this also provides a direction for how good approximations might be reached, as good approximation algorithms for knapsack exist.

A restriction of the problem where $r_i + p_i = d_i = \overline{d}_i$ reduces the problem to a directed acyclic graph (DAG) in which the optimal solution is the longest path. Interestingly, this makes the problem easily solvable, as the longest path in a DAG can be constructed in linear time. Two effects of changing the deadline thusly make the problem easier, the singular availability point ensures a job can only be scheduled at one time and cannot push back the completion time of later jobs. Secondly, if processing times are non-zero this means cycles of possible orderings cannot occur, simplifying the graph so a longest path algorithm can work.

Conversely, it is the pushing back of later jobs by scheduling another job earlier combined with the cycles in possible orderings of jobs that make OAS-SMS such a hard problem. The bounded width assumed in our special case resolves one of these issues, namely the cycles. The maximum number of jobs available to start at any one time point limits the number of job orderings that could be reversed and with it the number of cycles that require tracking.

3.1.2 Hardness proof

Here we will show that the bounded-width OAS-SMS problem is at least weakly NP-hard by showing that the 0/1 knapsack optimisation problem can be reduced to it, which itself is weakly NP-hard. The 0/1 knapsack problem is already trivially a subset of the full OAS-SMS problem; it is the special case where all setup times are zero, all release times are zero and the due dates and deadlines all are the same value. This encoding of knapsack as an OAS problem though is n wide, the proof below shows that it can also be encoded within the restriction of a constant-bounded width.

Theorem 1. *The 0/1 knapsack problem can be reduced in polynomial time to the bounded-width OAS-SMS problem when the width is bounded by a constant.*

Proof by reduction. Consider the 0/1 knapsack optimisation problem with items $J = 1...n$, weights w_j and revenue e_j for all $j \in J$, and some limit on the combined weight W.

The reduction is as follows: we add n jobs $b_1...b_n$ with processing time $p_b = W$, revenue $e_b = 1 + \sum_{j \in J} e_j$ and release times and deadlines $0...(n-1)W$ and $2W...(n+1)W$. The jobs

J are converted to have the same revenue and a processing time equal to their weight in the knapsack problem. They get release times $0...(n-1)W$ and deadlines $W...nW$. This results in at most 3 jobs have overlapping start time windows at any time point.

Jobs $b_1...b_{n-1}$ can all be scheduled in order, and each job b_i has a revenue that exceeds the total value of jobs $j \in J$ together. Therefore, any optimal solution must at least include all jobs $b_1...b_{n-1}$. For every job $j \in J$ there is a corresponding job b_i with the same release time. If j is scheduled, b_i must be scheduled after it or j could not complete before its deadline. For every job $j \in J$ that is scheduled, corresponding job b_i's start time is shifted by w_j, which in turn shifts the earliest possible start time of the next pair of jobs by w_j. If the summed weight of scheduled jobs from J exceeds W, some job b_i cannot be scheduled and the resulting schedule cannot be optimal.

Thus, an optimal solution to the OAS problem of value $nW + v$ corresponds to an optimal solution to the 0/1 knapsack problem of value v. Since the reduction takes polynomial time and the 0/1 knapsack optimisation problem is weakly NP-hard, the bounded-width OAS-SMS problem must be at least weakly NP-hard. □

The reduction from 0/1 knapsack shows the bounded-width OAS-SMS problem to be at least weakly NP-hard. Since the broader problem is strongly NP-hard, the question remains whether the bounded-width problem is too or not. As it turns out, it is not. There is a pseudo-polynomial algorithm for the bounded-width problem that iterates over the time points between the earliest possible start time of a job and the latest.

Let T be the latest possible start time for any job in a problem P, and let the earliest release time be 0, without loss of generality. Since release times, processing time and setup times are integers, T is a trivial upper bound on the number of time points on which a job in a schedule could start. Let $S_{t,j,V}$ denote a schedule where j is the last job scheduled and its start time is $t_j \leq t$, where V is the subset of all jobs that could start at t, denoted by J_t, which have been scheduled in S. Let $v_{t,j,V}$ denote the maximum value of all schedules with tuple (t, j, V) or $-\infty$ if no such schedule exists.

Clearly, the value at the source job 0 is $v_{0,0,\{0\}} = 0$. The following recurrence computes all values $v_{t,j,V}$ in $O(w2^w n^2 T)$ time, as there are T time points t, n jobs not including the source and sink jobs and at most 2^w different V for each t making for $O(2^w nT)$ values $v_{t,j,V}$, and each is calculated from n previous jobs for which V' takes $O(w)$ to calculate.

$$v_{t,j,V} = \max(v_{t-1,j,V \wedge V_{t-1}}, v'_{t,j,V}) \tag{3.1}$$

$$\begin{aligned} v'_{t,j,V} &= \begin{cases} \max_{j' \in J \wedge r_j + s_{j'j} \leq t}(v_{t',j',V'} + e_j - w_j \max(0, t - d_j))) & j \in J_t \\ -\infty & \text{otherwise} \end{cases} \\ \text{where } t' &= t - s_{j'j} - p_{j'}, \\ V' &= (V - \{j\}) \wedge J_{t'} \end{aligned} \tag{3.2}$$

The pseudopolynomial recurrence consists of two kinds of recurrence between which the maximum is taken in equation 3.1. The left side is the simple recurrence of assuming the job has been scheduled sometime earlier and waiting for a further time step. The second,

expanded in equation 3.2 considers the optimum way of scheduling job j to start exactly at t after coming from any other job j'. The maximum profit attainable for problem P is then found in the sink job $n+1$ as $v_{T,n+1,\{n+1\}}$. We thus have a pseudo-polynomial algorithm for the bounded-width OAS-SMS optimisation problem, so this bounded-width variant is weakly NP-hard.

3.2 Algorithms

Now that the complexity of the bounded-width OAS-SMS problem has been firmly established, we proceed to the construction of algorithms for it using decision diagrams. To do so, we first prove a number of properties of this problem that facilitate the construction of a decision diagram.

Recall that decision diagrams require an ordered set of decision variables and associatted domains that lead to all solutions as well as a state for the vertices to allow reduction during top-down construction. We begin with the development and proofs of valid decision variables and vertex states, solve the associated problem with identical states in different layers and then present an exact algorithm and a FPTAS, both based on decision diagrams.

3.2.1 Exact algorithm

Proposition 1. *Let π be a schedule for an OAS-SMS problem. There exists an* earliest-time schedule π' *which has equal or greater value and the same job ordering as π, for which the following two properties hold. No other schedule exists with the same jobs and job order where any job completes earlier than that same job does in π'. Secondly, the schedule π' can be constructed from the subset of jobs in it, ordered by their completion time.*

Proof by induction. If a schedule contains no jobs, there are no jobs which can be scheduled earlier or later and no construction is required, so the empty schedule trivially is an earliest-time schedule.

Given an earliest-time schedule S with completion times for some job order $[0,\ldots,i]$, which means job i is scheduled as early as it can be in any schedule with this job ordering. For any schedule with job order $[0,\ldots,i,j]$ the completion time of j must be at least $C_j \geq \max(C_i, r_j) + s_{ij} + p_j$ due to restriction 2.3. Consider the schedule S^+ where $C_j = \max(C_i, r_j) + s_{ij} + p_j$. All jobs preceding j are scheduled as early as possible, and given that they are so is j.

Thus, all jobs in S^+ are scheduled as early as they can be. The completion time can be iteratively calculated by $C_j = \max(C_i, r_j) + s_{ij} + p_j$. The value of a schedule is the summed value of all jobs in it $val(S^+) = \sum_{i in S^+} val(i, C_i)$ so $val(S^+) = val(S) + val(j, C_j)$. Because the value function for a job is non-increasing in its completion time, the value of a schedule must be greater or equal to one with the same job order but where jobs are completed later, so S^+ is also an earliest-time schedule. By induction propostion 1 holds for any earliest-time schedule thusly constructed. □

Proposition 1 states that for any schedule an earliest-time schedule exists with equal or greater value. This means that an algorithm for finding an optimal schedule can be limited

to finding the optimal earliest-time schedule represented by just a sequence of jobs instead of finding a sequence of job and start time tuples.

We proceed to create an exact algorithm for the bounded-width OAS-SMS problem through the use of a decision diagram. For a given problem a decision diagram requires a sequence of decision variables whose assignment fully defines solutions. The decision variables we will use is the decision of which job to schedule next, starting from no jobs scheduled and with the additional option of not scheduling any further jobs, as by proposition 1 such a sequence would encode all earliest-time schedules including an optimal one. Clearly, at most n jobs can be scheduled for a problem of size n, so there are $n+1$ decision variables $\{x_1,\dots,x_n\}$ where the domain of each decision variable is $D(x_i) = \{1,\dots,n+1\}$ with $n+1$ denoting not scheduling any further jobs, the sink job. The edges are labeled with the job and its discounted revenue so that any source-sink path has a cumulative value equal to the corresponding schedule's revenue.

A naive vertex state $S(v)$ for a vertex in the decision diagram would be the sequence of jobs scheduled so far. The decision diagram using such a state however would just be a decision tree. To represent the state more compactly, we use the fact that the problem width is limited. Any job in the sequence that is not currently available for execution is only relevant insofar as it contributes to a delay in the start time of the last job in the sequence. The last job is relevant because it influences setup times and so are any jobs in the past sequence that are still available at this time, as they cannot be scheduled twice. The state sufficient to encode all necessary information for determining whether subsequent schedules are feasible is formalised in the following lemma.

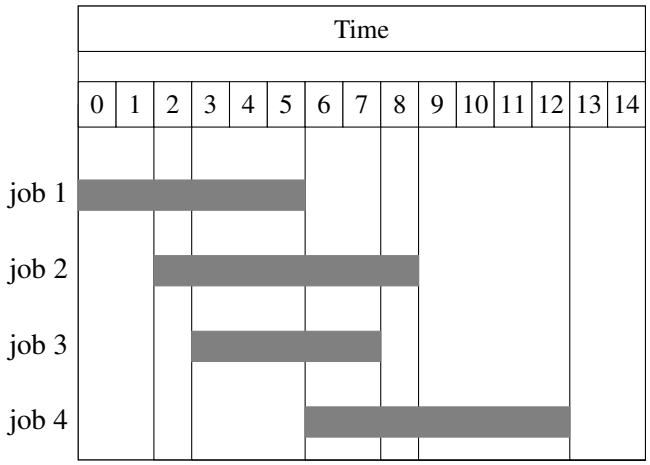
Lemma 1. *Let $S = [0,\dots,j]$ be a feasible partial solution with earliest start time for j being t_j and where S_v denotes those jobs in S which could be started at t_j, the* visited job state. *A schedule $S+S'$ where $S' = [k,\dots]$ is a feasible schedule if and only if S' is feasible for $t_k \geq \max(r_k, t_j + p_j) + s_{jk}$ and $S_v \wedge S' = \emptyset$. By extension, the tuple (j, t_j, S_v) is sufficient to determine the feasability of any subsequent scheduling of jobs.*

Direct proof. We first prove that S' being feasible for $\max(r_k, t_j + p_j) + s_{jk} \leq t_k$ and $S_v \wedge S' = \emptyset$ implies the feasibility of $S+S'$. There are three sufficient conditions for feasibility on a schedule given in the problem definition. For each job i, $t_i + p_i \leq \overline{d}_i$, for every pair of consecutive jobs h,i, $\max(r_i, t_h + p_h) + s_{hi} \leq t_i$ and the jobs in the the schedule must be a subset of all jobs, that is there must not be duplicates.

The first condition is met because S and S' are individually feasible, the second condition is met for all consecutive jobs that both lie in either S or S' for the same reason. The second condition for jobs (j,k) is met because t_k is defined identically to that condition. For all jobs $i \in S$, $t_i \leq t_j$ as that is the latest job in S. For all jobs $l \in S'$, $t_l \geq t_k$ as that is the first job in t_k. Since $t_k \geq t_j$ and all jobs can only be feasibly scheduled in their singular availability window, any job $i \in S$ which is also scheduled in S' must have an availability window that includes t_j. S_v contains all jobs scheduled in S whose availability windows include t_j, and $S_v \wedge S' = \emptyset$ so the third condition is met and $S+S'$ is a feasible schedule.

The reverse implication is proven as follows. The feasibility of $S+S'$ means that there is no overlap of jobs between S and S', $S \wedge S' = \emptyset$. Since $S_v \subseteq S$ by its definition, $S_v \wedge S' = \emptyset$.

Figure 3.1: The availability windows of jobs in the example problem from table 2.1, shown as horizontal bars with vertical lines denoting the borders between successive time windows. Note that for all time point in each time window the set of available jobs is the same.



Because $S + S'$ is feasible and jobs j and k are consecutive in it, j,k, the conditions on a feasible schedule give $\max(r_k, t_j + p_j) + s_{jk} \leq t_k$. □

Using these decision variables and vertex state, the optimal schedule for a bounded-width OAS-SMS instance can be found by constructing a decision diagram for it and finding the longest source-sink path, regardless of the start time or visited job state at the last vertex.

For easier reasoning about a decision diagram we use *time windows* for each problem instance. We define time windows here as those consecutive periods of time where the set of jobs J_W that could start is the same for all time points in that window. A new time window W thus is started at every time point t_W the availability window of a job starts or has ended the time point before. With n jobs, there are at most $2n+1$ time windows that together span all time.

Figure 3.1 shows how what the time windows would be for the example problem given in table 2.1. These time windows allow us to adapt one of the strong aspects of the pseudopolynomial algorithm, keeping track of what jobs could be started at time t in a set J_t. Instead of this being a $|T|$ long list of sets, only $O(n)$ sets for each time window need to be known. In turn, this allows us to use this broader definition of width than Balas and Simonetti [4] do. If we tracked jobs that could be swapped per job, job 2 would need to track all other jobs it overlaps with, being all 4, whereas time windows allow us to reduce this to the maximum overlap at any time point, which is 3.

Algorithm 1 shows the structure of the main part of the exact algorithm. In essence, it grows paths step by step by adding vertices until they all have reached the sink job. The order of the loops is only relevant in the sense that otherwise vertices for a specific job and visited job state could be processed before all vertices that match that have been created, and in that such an ordered partitioning where each edge leads to a later partition precludes cycles. This is why visited states in a time window are not processed by job first but by their

Algorithm 1 The top-down construction of an exact decision diagram for the bounded-width OAS-SMS problem using time windows and vertex state as per lemma 1.

```
for W = 1 ... |timeWindows| do                          ▷ O(n) such time windows
    for s = 1 ... |J_W| do                              ▷ O(w) such sizes
        for j ∈ J_W do                                  ▷ O(w) such jobs
            for S_v ∈ getVisitedJobStates(j, s, W) do   ▷ O(2^w) such visited states
                times = getTimes(j, S_v)
                successors = j' ∈ J where d̄_j' ≥ t_W
                for j' ∈ successors do                  ▷ O(n) such jobs
                    for t_j ∈ times do                  ▷ O(n!) or O(|T|) such states
                        if feasible((j, t_j, S_v) → j') then
                            u = addVertex((j, t_j, S_v), j')
                            resolveDomination(u)
return sink
```

window and size primarily. This guarantees groups of vertices are processed only after all vertices in that group have been created, as successor vertices can only lie in the same or a later time window, and if they lie in the same time window they must have a greater visited job state size as an additional job in that window has been scheduled.

In the implementation of the exact algorithm vertices are stored in dictionaries for each job so groups of them that share some property can be efficiently addressed through the getVisitedJobStates and getTimes methods. For each job j vertices are stored where j is the last job in the partial solution, and can be accessed in groups by their visited job state and time window. Jobs also maintain lists of visited job states for each number of jobs in such a state, and lists of potential successors for each window they are in consisting of all jobs that are available in their window or later ones. Windows have a list of the jobs in them, and jobs a reference to the time window they are in. The longest path's length and thereby highest total value to each vertex is saved in that vertex to save calculation time when considering domination rules. Feasibility is tested via constraints 2.2 and 2.3 and the next vertex and its state is calculated by the construction method in proposition 1.

Ultimately, for each vertex in the decision diagram so far and each potential successor job in the schedule it is tested if that successor job could be scheduled next without completing after its deadline. If it can be, the new vertex state is calculated and a vertex with that state is created if it does not exist already. An edge is then added with the successor job and the discounted revenue of scheduling it next as a label. The last step taken is to mark *dominated* vertices that do not need to be propagated, as described below. As in a branch and bound approach, it pays to remove dominated vertices during the construction of the decision diagram [15].

The absence of layers A striking difference between this exact decision diagram for the bounded-width OAS-SMS problem and the theory of decision diagram briefly summarised in chapter 2 is that there is no mention of *layers* in this formulation. Vertices are partitioned into ordered groups by time window and visited job state size, and top-down construction

proceeds group by group. These groups however are not associated with a specific decision variable, as schedules of different numbers of jobs can end in the same time window and visited size group. They do provide the same guarantee of preventing cycles and limiting the length of paths in the graph, as each edge from one group has to go to some later group, and a finite number of groups partition all vertices.

Instead, the vertices are simply a part of the unordered decision diagram and we use a certain partitioning of vertices to arrive at the same properties of the diagram, it is still acyclic and has a finite maximum path length. We call such a partitioning a *perspective* to denote its fundamental difference from the layers in previous work, multiple perspectives with each its own partitioning can be used for multiple purposes.

Runtime bound The runtime of the exact algorithm is bound by the number of vertices in the decision diagram created and the time it takes to calculate them. Each vertex has a unique state (j, t_j, S_v). $O(n)$ time windows which each have $O(w)$ jobs that could start there with at most $O(2^w)$ different visited job states. It is the different options for t_j that prevent a polynomial exact solution for the bounded-width OAS-SMS problem, as every preceeding sequence of jobs could result in a different start time t_j, for an $O(n!)$ or $O(|T|)$ bound. The approximation scheme presented in the next section will resolve this potential growth in t_j to achieve a polynomial runtime bound. The work in calculating a state is linear in w as a new visited job state has to be calculated as $S'_v = S_v \wedge \text{window.jobs} \vee \{\text{successor}\}$, while the successor job is already known and its start time is calculated in constant time given the start time of the previous job. The runtime bound of the exact solution therefore is $O(nw^2 2^w n!)$ or $O(nw^2 2^w |T|)$.

Domination After a new vertex is created, vertices that might be dominated are checked and marked if they are dominated by this new vertex. Conversely, if the new vertex is dominated, it is marked as such. Dominated vertices have equal or fewer subsequent schedules available and shorter paths to them, which guarantees that the vertex that dominates them can reach at least as good a schedule as they could. This is formalised in the following proposition.

Proposition 2. *For any two vertices a and b in the decision diagram with best accumulated values v_a and v_b, if $t_a \leq t_b$, $v_a \geq v_b$, $j_a = j_b$ and $S_a \subseteq S_b$, then if there is an optimal sequence containing b there is also an optimal sequence containing a.*

The proof is trivial, vertex a puts fewer restrictions than b on subsequent sequences. We say that *a dominates b*, written as $a \succ b$. Vertices that are dominated can be skipped in the exact algorithm, as all possible subsequent sequences are covered by some other vertex to greater ultimate value. This leaves us with for each job j and visited job state S_v only a *pareto front* of vertices where no vertex has both later start time and lower accumulated value than any other vertex.

3.2.2 A fully polynomial time approximation scheme

The exact algorithm constructed in the section above is exponential in n due to the potential growth of start times for states with a given job and visited job state. The dominance relations ensure that no pair with both lower accumulated value and later start time is propagated, but this does not mitigate the exponential growth of these in the worst case, as there can be infinitely many non-dominated points on a pareto front.

This no longer holds if we allow a small loss of value. In that case, vertices with higher value but a later start time can be merged into one with lower value and an earlier completion time. The earlier start time guarantees that at least as many subsequent schedules are available, and the maximum value lost is the difference between the value of the vertex that is kept and the highest value vertex merged into it. A general proof that a relaxed decision diagram can be constructed in this way for a decision diagram without layers is given in appendix A. Here, the thing that prevents cycles and infinite paths is that only states with equal last job, time window and visited state size are merged and result in a state that has those same job, time window and visited state size, while edges must always lead to a state in a strictly later such partition.

The remaining vertices approximate the pareto front as a lower bound by tracking the difference in value, with the highest value vertex merged into each vertex remaining an upper bound can also be achieved. The following lemma proves this in general for any pareto front of time and value pairs.

Lemma 2. *Given a non-empty set P of x pairs of positive real values (t, v), where there are no two distinct pairs $i, j \in P$ such that $t_i = t_j$ nor such that $t_i < t_j \wedge v_i \geq v_j$, that is the pairs form a pareto front. Let a be some real positive value. Let v_+ be the greatest value of v for all pairs in P and v_- the smallest.*

There exists a subset $P_a \subseteq P$ of at most $\left\lceil \frac{v_+ - v_-}{a} \right\rceil$ pairs such that for each point $i \in P$ there is a point $j \in P_a$ where $t_j \leq t_i$ and $v_j + a \geq v_i$, noted as $j \succ i$. This subset can be constructed in polynomial time with respect to $|P|$.

Proof. Let P be such a set and a such a value. We will construct a subset P_a, starting from an empty set.Consider the set P_r consisting of points i for which there is not yet a point j in P_a such that $j \succ i$, with $v_{r,+}$ and $v_{r,-}$ the greatest and smallest values of v in P_r.

Now repeatedly the pair $j = (t, v_{r,-}) \in P_r$ is added to P_a. By the properties of all pairs in P this is also the lowest t in P_r, so every job $i \in P_r$ for which $v_j + a \geq v_i$. Each such pair j covers all pairs in a range of size a in $[v_-, v_+]$ except the last one added which only covers its own v up to v_+ and these ranges do not overlap, so at most $\left\lceil \frac{v_+ - v_-}{a} \right\rceil$ such are added to P_a. □

Theorem 2. *If an approximation factor ε is chosen such that any solution σ with $v(\sigma) \geq (1-\varepsilon)\pi$ is acceptable, then the number of (time, value) pairs in each decision diagram vertex for a given current job and visited job state can be limited to a polynomial number in polynomial time in n and $1/\varepsilon$ without losing the guarantee to find an acceptable solution.*

Proof. Let π be an optimal sequence and ε the approximation factor. No sequence can consist of more than n jobs, so any source-sink path in the decision diagram cannot contain more than n vertices, excluding the source and sink.

By proposition 2, for any vertex with earlier completion time can reach at least as many subsequent paths as any other vertex with the same current job and visited job state.

For each combination of current job and job state, we reduce the pareto front to at most $\left\lceil \frac{v_+ - v_-}{\varepsilon v_+} n \right\rceil \leq \frac{n}{\varepsilon}$ vertices such that for every job originally in that pareto front there is one within $\frac{\varepsilon v_+}{n}$, as is proven possible by lemma 2.

Since there are at most n jobs in a sequence, at most $\frac{\varepsilon v_+}{n} \leq \frac{\varepsilon \pi}{n} n = \varepsilon \pi$ value is lost for each schedule in this restricted decision diagram. Since this is done in polynomial time, a sequence within ε of π will be found in polynomial time. □

Runtime bound Due to the approximation ratio, the number of vertices with the same job and visited job state is now limited to $O(\frac{n}{\varepsilon})$. This gives a bound on the number of vertices of $O(\frac{w2^w n^2}{\varepsilon})$, calculating state remains the same, but maintaining a representive list involves a linear time insertion procedure for each new vertex. The FPTAS therefore has a runtime bound of $O(\frac{w2^w n^3}{\varepsilon^2})$.

3.2.3 Algorithm refinements

Some additions to the algorithms above can be made that can reduce the running time of the algorithm greatly for practical applications, though they do not improve on the theoretical worst case complexity. Some are computationally heavy enough that they may increase the worst case complexity.

Distant successors Consider jobs j and k and some time window W_t. If r_k is sufficiently large that for any path where j is started in W_t, $C_j \leq r_k$, then any sequence with j started in W_t that has k as the next job will start at the same time with k. This means that for each different visited state, only the largest value counts when it comes to distant successors like k. The other paths would have been dominated at k anyway, but this saves the calculation of the new path points and domination checks, and all tuples (j, W_t, k) can be precomputed.

Mandatory jobs If at some point between two jobs in a schedule we pick one job, but another would lead to a better value or completion time if inserted in that slot, then that other job must be included later in the sequence or it would have been better to insert here. This is more precisely formulated in the lemma below.

Lemma 3. *Let a and b be vertices in the decision diagram where $t_a \leq t_b$, $v_a \geq v_b$, $j_a = j_b$ and $S_a - S_b = \{j_x\}$. If any optimal solution π contains b there is one that contains job j_x.*

Proof by contradiction. Let a and b be vertices in the decision diagram where $t_a \leq t_b$, $v_a \geq v_b$, $j_a = j_b$ and $S_a - S_b = \{j_x\}$. Assume there is optimal solution π that includes b but there are none that include j_x. Because π does not contain j_x, we can restrict the problem without losing the optimal solution π, by reducing the availability of j_x such that it can no longer

be started on or after t_a. In this restricted problem, vertex a is still feasible but $S_a \subseteq S_b$. By proposition 2, there must be an optimal sequence that includes a and thereby j_x. This is a contradiction, therefore lemma 3 is true. □

This is only of interest to limit the $O(2^w)$ visited state growth somewhat, as each time the states are equal again, regular domination by proposition 2 would suffice. Where the width is not close to a limiting factor, the extra computation likely weighs heavier.

3.3 Applicability to other problems

The OAS-SMS problem is a very general problem that contains as special cases many other well-known problems such as the 0/1 knapsack problem and many variants of travelling salesman and travelling repairman. Additionally, concepts used here in developing algorithms for the OAS-SMS problem, specifically those concepts used in the decision diagrams, have broader applicability, some of which are detailed below.

There are a number of differences in the way decision diagrams are used in this thesis versus how they are used canonically [5]. The most fundamental change is a new interpretation of the concept of *layers*.

The two other changes can be briefly summarised as they are relatively small additions. One is that some space may be saved when creating both a relaxed and a restricted decision diagram by putting all vertices of them in one combined diagram and labeling relaxed and restricted vertices as such. This allows any common exact vertices to be shared and the best path can be calculated for each separate diagram by ignoring those vertices associated with the other.

The removal of dominated vertices is not new [15], but to do so the state of the dominating vertex must not only be a relaxation of the dominated vertex but also have a longer or shorter path to it from the source vertex in maximisation and minimisation problems respectively. During top-down construction of a decision diagram this results in a repeated execution of the longest path algorithm, which is inefficient. Maintaining in each vertex that will not receive any new incoming edges the value of the best path to it saves calculation time, though at the cost of a constant space increase per vertex.

Reimagining layers In section 2.2 where we laid out the basic structure of a decision diagram we associated each vertex with a layer, and associated each layer with a decision variable. This makes decision diagrams neatly organised and easy to understand, each edge takes one to the next layer by deciding on that layer's associated decision variable, so in layer L_0 no decisions have been made and in layer L_n all have been decided.

Some deviation from this practice already exists in the form of *long edges* that span multiple layers. These are created by removing vertices with only one outgoing edge and letting their incoming edges connect directly to their one successor vertex. Each vertex however is still firmly placed in a layer associated with the last decided variable, and only vertices in the same layer are considered for merging in reducing the diagram or for relaxation or restriction purposes [19].

In the algorithms described above, a solution is represented by a schedule, which can be condensed as a permutation of a subset of the jobs that make up a problem instance. There are several ways to encode this as decision variables, we used the decision of what job to schedule next, with one added option of not scheduling any further jobs. A classical decision diagram would then place each vertex into a layer associated with the number of jobs scheduled so far.

This does not make sense though when considering the conditions for merging vertices. The set of possible subsequent schedules does not depend on the number of jobs scheduled so far at all, instead it depends on the time, the last job and the jobs still available. Nor is there an easy bound on the number of states where m jobs have been scheduled. To solve this, we do not use layers at all, but keep different *perspectives* on the vertices for different purposes.

To get a runtime bound for the problem we use the perspective of the *time windows* in which the same jobs are available so we can use the constant width w to limit the visited job states possible. To prove the diagram is acyclic, we use both the time windows and the size of the visited job state together as a perspective as mentioned in the explanation of the exact decision diagram algorithm. Vertices are merged or dominated through a number of different perspectives.

3.3.1 Generalising the Balas-Simonetti neighbourhood

A similar but less general width concept is used in travelling salesman problems to explore an exponential neighbourhood of solutions in time that is linear in the problem size but exponential in a width analogue defined by a constant k. This is the Balas-Simonetti neighbourhood [3, 4]. Below we give a brief definition of this neighbourhood and give some examples of how it is used in current state of the art. Then we show that the width analogue they use for a constant k is a special case of the width used in this thesis with $w = k$, and that the same runtime bound that applies to their special case applies to the more general case of problems restricted by the width w.

For a general travelling salesman problem, a Balas-Simonetti neighbourhood is defined by an ordering of the cities $\{1,\dots,n\}$ and a constant k. A solution given by a permutation π lies in this neighbourhood if and only if for every two jobs i and j where $i+k \leq j$ the permutation is such that $\pi(i) < \pi(j)$. The constant k thus defines a distance in the ordering $\{1,\dots,n\}$ where any two cities that are that far or farther apart in the ordering must be visited in the order they have with respect to each other in that ordering. The optimal sequence in the neighbourhood can then be found in $O(k^2 2^{k-2} n)$ time [3]. They also give a generalisation where constants $k(i)$ depend on the city, but the runtime bound on that variant is not, in general, better than that for constant k where $k = \max_i(k(i))$.

Because of its linear algorithm and the exponential number of solutions [11] it searches exhaustively, the Balas-Simonetti neighbourhood is used for local search [12]. It efficiently finds the optimal solution in the neighbourhood, and provides a lower or upper bound for the overall maximisation or minimisation problem respectively. Bergman et al. also briefly touch on this neighbourhood in their discussion of the applicability of decision diagrams to single machine scheduling [5]. As this neighbourhood is apparently still in use, a generali-

sation of it with the same runtime bound could be of use, though it will not be the focus of the remainder of this thesis.

The width used in the algorithms discussed in previous sections is defined for each time window as the number of jobs that could be scheduled in that window, based on their availability windows. To adapt this to the travelling salesman problem a slight modification is required as it does not have time windows. Instead the availability window of a city is the range of consecutive positions it could have in solutions. The width for a position is the number of jobs that could have that position in a solution in this neighbourhood, and the width of the problem is the maximum such width. A Balas-Simonetti neighbourhood of width k is a subclass of a bounded-width neighbourhood of width $w = k$, which follows directly by examining the width at each position.

Proof. Let $J = \{1,\ldots,n\}$ be a TSP problem of n cities. Let $1,2,\ldots,n$ be an ordering of cities for which k defines a Balas-Simonetti neighbourhood. We will construct a decision diagram for this problem where the decision variables are the city picked for each position in the tour. The state used is a tuple of the first city not yet picked, the available cities already picked and the last city picked (i,S_v,l). The *perspective* used, analogous to time windows in the algorithms above, is the first city that has not yet been visited. Consider any point at which city i is the first city in the ordering not yet visited. The k neighbourhood means city $i+k$ and beyond cannot be visited next, so there are at most $k-2$ cities that could be visited next but also might already have been visited. □

The last city picked similarly cannot be any city beyond $i+k-1$ nor any city before $i-k+1$, and there are n possible first cities that have not been picked yet. This limits the vertices to $O(k2^{k-2}n)$ states, and since up to k cities are available as the next city for any state, there are $O(k^2 2^{k-2}n)$ edges. The visited state S'_v for a next job takes $O(k)$ time to compute, but is the same regardless of the previous last job and the rest of the state can be constructed in constant time, so the algorithm as a decision diagram implementation also takes $O(k^2 2^{k-2}n)$ time.

Chapter 4

Datasets

We present a number of datasets in this chapter to evaluate the exact algorithm and the FPTAS against. These evaluations will have two purposes. Firstly, the performance of these algorithms should be evaluated against the state of the art OAS-SMS algorithms, to see if a focus on bounded-width problems results in better performance on those instances. Secondly the difficult cases for the exact algorithm where the FPTAS may outperform it should be explored to evaluate in what cases the approximation scheme improves performance and by how much.

4.1 OAS-SMS Dataset

The first dataset used to evaluate the exact algorithm is the one created in [8]. They generated it in the same way though for some different parameter settings as [22]. It provides a useful benchmark, as subsequent papers like [8], [21], [20] and [24] all also use it to compare their improvements against the 2010 paper and each other.

The dataset consists of 1500 randomly generated instances, with 3 parameters and 10 different instances per combination of parameters. The first of these parameters is the number of jobs n, ranging from 10 to 100. The other two parameters have multiple effects. The second parameter τ governs how spread out over time the release dates are, as well as how small the availability windows are. A larger value for τ thus corresponds to availability windows that are both more spread out and are smaller, which stochastically reduces the width of the problem, making them suitable for the algorithms in this paper. The last parameter R governs how variable the sizes of the availability windows are and how far beyond the due date the deadlines lie. The latter increases the width, though less strongly so than does τ.

For a given parameter tuple (n, τ, R) the 10 instances are created as follows. The processing time, value, setup times, release date and *slack* for each job were picked randomly from the integer ranges $[1,20]$, $[1,20]$, $[1,10]$, $[0, \tau p_T]$ and $[p_T(1-\tau-R/2), p_T(1-\tau+R/2)]$, where $p_T = \sum_{j=1}^{n} p_j$. Since $\tau < 1$ not all jobs can be scheduled. Due dates and deadlines are calculated from the equations $d_i = r_i + \max_{j \in J}(s_{ji}) + \max(\text{slack}_i, p_i)$ and $\overline{d}_i = d_i + Rp_i$. The weights were determined by their relation to value, deadlines and due dates in the definition of the OAS-SMS problem, $w_i = e_i/(\overline{d}_i - d_i)$.

Due to the way the slack in the availability windows is generated, all jobs can be scheduled after any other based on their setup time from the previous job. This is not actually included in the definition and does change the problem qualitatively. Instances where setup times can be so large between jobs that some orderings are infeasible, even if their availability windows would allow it, could comprise a kind of limited precedence constraint structure.

4.1.1 Limitations

The dataset has many limitations to how much of the entire class of OAS-SMS problems it covers. The jobs are generated on uniform distributions for their release time, due dates and deadlines, value and setup times.

The independent uniform distributions of processing time and value lead to instances where there is a clear partial preference ordering by value over processing time. The scaling of values between the lowest job value and the highest is also limited and the scaling of processing time and setup times is equally limited leading to a limited set of value and processing time combinations jobs can have. In various subproblems like $1||\sum wU$ these kind of scalings allow for pseudopolynomial algorithms [16]. It is not clear whether these affect the problem without width restrictions.

The variance in the value to processing time ratio of the jobs is of particular interest to the performance of algorithm 1. This exact algorithm becomes exponential only if the number of non-dominated (time, value) pairs grows exponentially. Intuitively this requires that the value to processing time ratio should be similar for many jobs. As a result, the difference between the exact and approximation algorithm might not show for instances in this dataset.

The integer intervals from which the processing time, revenue, setup time and by extension the availability windows are chosen are very small. As shown in the complexity analysis, there is a pseudopolynomial solution to the bounded-width problem if all time points can be iterated over. In this dataset the mean processing time is $p_{avg} = 10.5$, so even for $n = 100$ the mean total processing time is only $p_T = 1050$. While the exact and approximation algorithm do not explicitely use iteration over time points, there are likely to be collisions where different vertices in the decision diagram have the same time without being otherwise related paths. This unnecessarily complicates evaluation of algorithms that solve it. An algorithm could just be good, or the performance could be thanks to a large number of collisions that might not ever occur in real world applications.

4.2 Additional bounded-width dataset

The limitations of the dataset generation procedure make it quite unlikely that instances in it will show off the exponential growth of vertices in the decision diagram. Additionally, because width was not a consideration when the dataset was constructed, the width can vary greatly for fixed parameters (n, τ, R) and is positively correlated with each of them. A new dataset is generated with two-fold purpose: to set a constant width so its influence on the

run time of the algorithms is constant, and to create instances that lead to an exponential growth in decision diagram vertices for the exact algorithm.

The instances are identified by three parameters. The number of jobs n, the width w and the due date range R. The parameter R here is similar but slightly different than in the existing dataset, because in that dataset it affected the width. The width parameter w sets a constant width for the problem for all time points between the source and the sink jobs except near the sink where the last jobs can become unavailable one by one.

Instances were generated with the following values for each parameter, $n = 20, 40, 60, \ldots, 200$ jobs, $R = 0.1, 0.5, 0.9$ and $w = 6$. Only a single value for the width was used as the purpose of this dataset is to explore the effect of just n and R on the runtime. The width 6 was chosen because it is small but not so small that there is likely to be a special case solution for the problem of this width, which there might be for say $w = 2$.

One of the purposes of this dataset is that the number of vertices in the exact decision diagram might grow close to exponentially for increasing n. Since this depends on a large number of time values t being available for each state, the availability windows and other time values should not be from such a small interval as $[1, 20]$. Instead, a scale of 2^{40} is chosen for these values. In the existing dataset processing time, revenue and setup time are chosen on $[1, 10]$ or $[1, 20]$ intervals, which we consider to be a scale of 10. This scale of 2^{40} is chosen here so it lies between two values. It should be high enough that iteration over time points is infeasible and collisions unlikely. It should also be low enough that for any combination of jobs in a schedule the total time and value are not close to exceeding the 64-bit integer size.

The instances were generated in the following manner. Processing times were generated at random from a uniform distribution over the integer range $[1, 2\text{scale}]$, revenue from real range $[p_i - \text{scale}/n, p_i + \text{scale}/n]$ for each job i and setup times from the integer range $[1, \text{scale}/n]$. This way the value is strongly correlated with the processing time and there are few occurences of jobs that are strictly better than other jobs in terms of processing time and revenue.

The difference between the release time and the deadline is uniformly at random generated from the integer range $[p_i + \frac{1}{2}\text{scale}(w - wR) + \text{scale}, p_i + \frac{1}{2}\text{scale}(w + wR) - \text{scale}]$. The average slack of $w\frac{1}{2}\text{scale}$ is chosen so that feasible schedules can contain about half the jobs, the minimum slack is such that setup times never prohibit an ordering on their own and R has the same role it does in the existing dataset. The due dates are set to $\overline{d}_i - Rp_i$ for each job i, and the penalty weight such that the value becomes zero at the deadlines.

4.3 Correlation effect dataset

A subclass of the 0/1 knapsack problem is the Subset Sum problem, where the value of each item is perfectly correlated with its weight. Perhaps surprisingly this can make the problem harder, one might expect subsets with an item of large weight and low value to be dominated by one that picked higher value per weight items, but in subset sum no such domination occurs as each subset's value and weight lie perfectly on a pareto front consisting of a line.

This seems similar to the worst case of the exact decision diagram for the bounded-width OAS-SMS problem, so we created a dataset of instances where the correlation between the value and processing time of a job increases from no correlation to perfect correlation and simultaneously the setup times become zero to make it even closer to a subset sum problem.

Specifically, in this dataset for a choice of correlation parameter c the setup times s_{ij} are uniformly randomly picked from $[1, \text{scale}(1-c)]$ and value e_i from the real range $[cp_i + (1-c), cp_i + (1-c)\text{scale}]$. The effect here is that for $c = 0$ there is no correlation between processing time and revenue, much like in the OAS-SMS dataset, for $c = 1 - \frac{1}{n}$ the correlation is like the bounded width dataset and for $c = 1$ the time to process a job is exactly its value like in Subset Sum and setup times are zero.

The width is set to be a constant, w, except at the end of the problem where the availability windows of the last few jobs do not all end at the same time. This is achieved by setting the release time r_i of each next job $1 \dots n$ at the first time point where the width is not yet w. The slack and thereby due date and deadline for each job are calculated exactly as in the bounded-width dataset in section 4.2, so that similar to there about half the jobs can be scheduled.

Chapter 5

Results

In this chapter we evaluate the exact algorithm and the FPTAS on the dataset presented in chapter 4. The results on the OAS-SMS dataset is used to compare the performance of these algorithms for problems of bounded width against state of the art algorithms for the general OAS-SMS problem. To this end we only consider instances in that dataset with parameter $\tau = 0.9$, as these are the instances where the width is relatively small compared to the number of jobs n. Due to the exponential dependence on width of the runtime of the presented algorithms we expect to see a cutoff width, where for smaller widths the presented should algorithms exceed the state of the art.

We also aim to evaluate the difference between the exact algorithm, with its exponential worst case time complexity, and the FPTAS. From an analysis of the OAS-SMS dataset we expect that dataset to be of limited use in this endeavour, because it has no correlation between job processing time and value. The bounded-width dataset presented in section 4.2 was created to be similar to the existing OAS-SMS dataset, except in that its width is a parameter and processing time and value are correlated to some extend. This dataset is used to compare the exact and approximate algorithms against each other, we expect to see a qualitative difference in performance here, as the increased correlation should in turn lead to vertex states having increased correlation between their start time and the longest path to them.

The question of how correlation between job processing time and value affects the difference in performance between the exact and approximate algorithms is further explored using two other datasets, one presented in sections 4.3 having correlation as a parameter of the instances in it, and the other an encoding of Subset Sum instances in the manner of the knapsack reduction presented early in chapter 3. We expect the Subset Sum instances to most clearly show the exponential runtime growth versus problem size of the exact algorithm, whereas for the FPTAS it should only grow polynomially. We expect the dataset where correlation is a parameter to show a sharp increase in CPU time when there is near perfect correlation between processing time and value.

The exact and approximation algorithms were implemented in native Python 3.7. Experiments were run on a 2GHz Intel Core i5 with 8GB RAM running macOS 10.14, in a single thread. Each group in the dataset given by [8] of 10 instances with the same parameter tuple (n, τ, R) was given a collective time limit of 10 hours. Where runtime exceeded

10 hours for a group, no results are reported. The minimum, average and maximum time taken for finding the optimal solution for instances in each group are saved. Because the algorithms have no random components, each instance is solved only once.

5.1 OAS-SMS Dataset

The minimum, average and maximum CPU time for the exact algorithm, on each group of 10 instances for which the parameters (N, τ, R) are equal and $\tau = 0.9$, is shown in table 5.1. For the approximation algorithm with $\varepsilon = 0.1$, these are shown in table 5.2. For both some values are missing for $n = 100$, for these the execution time of the algorithm on the group of 10 instances exceeded 10 hours.

The approximation rate $\varepsilon = 0.1$ was chosen for two reasons. The previously best known upper and lower bounds for the $n = 100$ instances, found by Silva et al. [24], differ by around 10% so to achieve competitive bounds ε should not be much larger. If ε were chosen much smaller, very few vertices would be merged because the values used for the properties of the jobs in these instances are such that there can only be in the order of 10 different start time and value combinations.

The CPU time versus the width of the problem instances are shown in figure 5.1. The exponential nature of the exact algorithm in w is clearly visible. Because width is not a parameter of the OAS-SMS dataset, instances with the same parameters can have widths that differ. As can be seen in this figure, this can greatly affect the CPU time and explain the large variances seen in tables 5.1 and 5.2.

5.1.1 Analysis and comparison with literature

The CPU time values in table 5.1 increase strongly for larger n, and also, though less strongly so, with increasing R. There is a lot of variance among instances with the same parameters as well. This is likely due to how much variance there is in the width of these instances of which the effect can be seen in figure 5.1. As that figure shows, there also is some variance in CPU time among jobs of equal (maximum) width. One cause of this may be that the maximum width provides an upper bound on the number of possible visited job states, but in specific instances this number can be much smaller. Consider a problem where w jobs are available at some time t but at no other time. Then instead of 2^w visited job states there are only w as only one of the w jobs can be scheduled.

As discussed earlier, both τ and R are correlated with the width, but the random release times of jobs can lead to very wide sections in some but not other instances of the same parameters. In figure 5.1 this is especially apparant for the $n = 50$ instances. The CPU time depends much more strongly on width than on problem size, which is promising for the ability to solve even larger problems, provided they have a limited width. For those instances where the width was amenable, the exact algorithm clearly outperformed the previous exact solutions in [8] and [24]. Optimal solutions were found for some $n = 100$ instances, none of which had previously been solved to optimality within a time limit of one hour, and on less powerful hardware. For the problems of smaller width they were solved in far less time than an hour, these should no longer be considered difficult problems

Table 5.1: Aggregate exact results for $\tau = 0.9$. CPU time results are reported for each group of 10 instances, only if the average time was under 1 hour. The CPU times are rounded to two significant digits.

n	R	Time (s) $\tau = 0.9$		
		min	avg	max
10	0.1	< 0.01	< 0.01	< 0.01
	0.3	< 0.01	< 0.01	< 0.01
	0.5	< 0.01	< 0.01	< 0.01
	0.7	< 0.01	< 0.01	< 0.01
	0.9	< 0.01	< 0.01	0.013
15	0.1	< 0.01	< 0.01	< 0.01
	0.3	< 0.01	< 0.01	0.011
	0.5	< 0.01	0.013	0.045
	0.7	< 0.01	0.018	0.069
	0.9	< 0.01	0.031	0.127
20	0.1	< 0.01	< 0.01	0.018
	0.3	< 0.01	0.016	0.032
	0.5	0.014	0.036	0.11
	0.7	0.024	0.050	0.13
	0.9	0.054	0.14	0.50
25	0.1	0.014	0.021	0.036
	0.3	0.017	0.036	0.073
	0.5	0.041	0.098	0.23
	0.7	0.060	0.31	1.3
	0.9	0.049	0.35	1.5
50	0.1	1.0	1.3	1.8
	0.3	1.0	2.3	3.4
	0.5	2.8	11	55
	0.7	4.0	35	150
	0.9	4.7	350	2000
100	0.1	170	500	1400
	0.3	200	1300	4300
	0.5	–	–	–
	0.7	–	–	–
	0.9	–	–	–

Table 5.2: Aggregate approximate results for $\tau = 0.9$ with an approximation rate $\varepsilon = 0.1$. CPU time results are reported for each group of 10 instances, only if the average time was under 1 hour, and are rounded to two significan digits

n	R	Time (s) $\tau = 0.9$		
		min	avg	max
10	0.1	0.12	0.14	0.17
	0.3	0.12	0.13	0.15
	0.5	0.11	0.12	0.13
	0.7	0.12	0.13	0.14
	0.9	0.11	0.13	0.14
15	0.1	0.17	0.18	0.19
	0.3	0.17	0.19	0.20
	0.5	0.17	0.19	0.23
	0.7	0.18	0.19	0.23
	0.9	0.19	0.21	0.30
20	0.1	0.24	0.29	0.43
	0.3	0.21	0.26	0.29
	0.5	0.25	0.28	0.36
	0.7	0.27	0.30	0.40
	0.9	0.29	0.36	0.68
25	0.1	0.32	0.34	0.38
	0.3	0.31	0.35	0.39
	0.5	0.34	0.41	0.59
	0.7	0.47	0.74	1.9
	0.9	0.36	0.73	1.9
50	0.1	1.4	1.9	2.9
	0.3	1.2	3.2	5.1
	0.5	3.9	14	53
	0.7	5.7	52	210
	0.9	5.6	420	2200
100	0.1	320	1000	2900
	0.3	910	2900	9100
	0.5	-	-	-
	0.7	-	-	-
	0.9	-	-	-

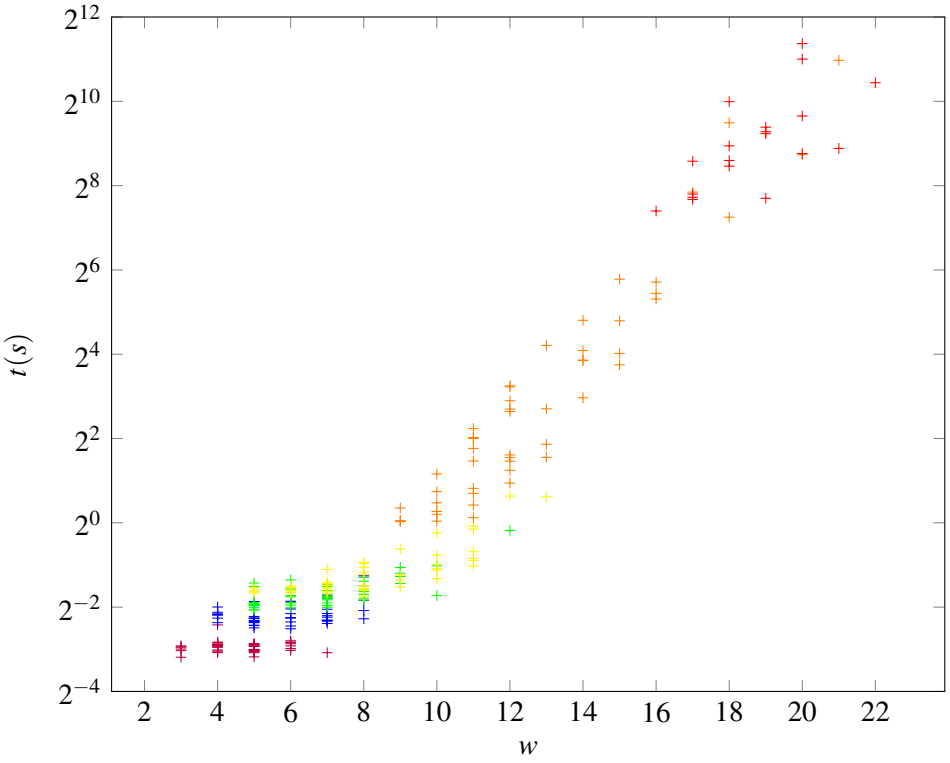


Figure 5.1: CPU time versus width w for the exact algorithm on the OAS-SMS dataset for $\tau = 0.9$ and with only those instances of $n = 100$ where solutions for all 10 with the same parameters were found in 10 hours. The problem size $n = 10, 15, 20, 25, 50, 100$ is shown in the colours purple, blue, green, yellow, orange and red respectively.

One striking result is that the FPTAS in almost all cases performed worse than the exact algorithm in terms of CPU time. This is not as surprising as it may seem. The FPTAS merges vertices in its decision diagram which have the same last job j and visited job state S_v and should thereby reduce the number of vertices in the diagram. However, we do this both to create restricted as well as relaxed vertices, so more than two vertices need to be merged before there is an actual reduction in the number of vertices and creating these merged vertices also takes time. Furthermore, the lack of correlation between job processing time and value causes many paths to be dominated by other paths, these would already be removed in the exact decision diagram. Thus the instances might not actually result in such a diagram as would be meaningfully reduced in size by the FPTAS, moreso than the extra time it takes to calculate merged vertices.

5.2 Bounded-width dataset

The CPU time versus problem size n for the exact and approximation algorithm with $\varepsilon = 0.1$ are shown in figures 5.2 and 5.3. In both graphs the results for instances with the same parameters are displayed as the mean with above and below it a shaded area up to one standard deviation away. For the three different $R = 0.1, 0.5, 0.9$ the colours blue, green and red are used respectively. While the exact algorithm does not definitively display exponential behaviour, the CPU time for it increases much faster than it does for the approximation algorithm.

For instances of size greater than $n = 140$ the CPU time of the exact algorithm becomes prohibitive. In figure 5.3 the approximation algorithm for $n = 120$ and $R = 0.1$ shows a bump compared to the CPU time of surrounding n. This is not due to a single outlier, multiple instances with these parameters took longer. These instances were solved a second time, where they had a lower mean CPU time and variance, so the increased CPU time for these instances is likely due to some other process on the test computer taking up CPU time or memory during the solving of these instances.

5.2.1 Subset Sum and correlation dataset

For an encoding of the Subset Sum problem as a bounded-width OAS-SMS problem, done in the manner as described for encoding a 0/1 knapsack problem, the CPU time is shown in figure 5.4. The exact algorithm's CPU time is shown in blue, that of the approximation algorithm in red. The exact algorithm was run on instances up to $n = 30$ due to the large growth of CPU time for small increases in size. From this point the approximation algorithm was only run on instances of sizes divisible by 10.

The CPU time for the bounded-width dataset with various values for c, a parameter that governs the amount of correlaation between job processing time and revenue, is shown in figure 5.5. More instances were used close to $c = 1$ for the exact algorithm, the results of which are shown in blue. The results for the approximation algorithm is shown in red.

5.2.2 Analysis

Figure 5.2 shows a rapid growth in CPU time for the exact algorithm on problems of increasing number of jobs n. The CPU time does not increase exponentially as indicated by the decreasing slope of the curves, at least for $n \leq 60$. This indicates that the bounded-width dataset does not consist of perfect worst case instances for each size, which is not unexpected as this would require that there not be an increasing fraction of dominated partial solutions for increasing n. Above $n = 60$ it is not clear if the CPU time does or does not increase exponentially, but it does grow rapidly, taking close to an hour for instances where $(n, R) = (140, 0.1)$.

For all n, there is a reduction in CPU time for increasing the due date range R. This can be explained intuitively, earlier due dates effectively shrink the time window in which a job is cost effective in the value per processing time sense. Even though the availability windows and width do not change for increasing R, some jobs might not be cost effective outside a very small time window.

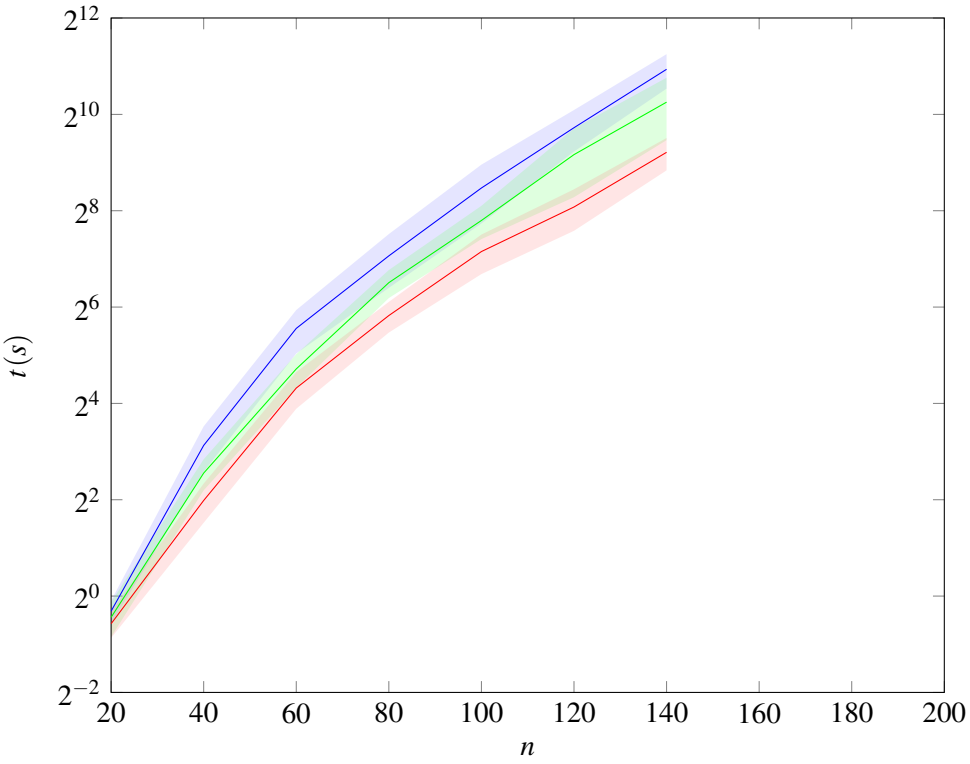


Figure 5.2: CPU time versus number of jobs n for the exact algorithm on the bounded width dataset for instances up to $n = 140$ in size. Results for $R = 0.1, 0.5, 0.9$ are shown in blue, green and red respectively. This dataset has 10 instances for each (N, R) for instances up to $n = 100$, for $n = 120$ and $n = 140$ only 3 instances were used due to prohibitive CPU time for each. A line shows the mean time and an area around it up to one standard deviation away is shaded.

Even though the CPU time of the exact algorithm does not grow exponentially in n as it would in the worst case, the approximation algorithm's CPU time grows much slower as can be seen in 5.3. A similar reduction in CPU time for increased R is seen here as with the exact results, likely for the same reason.

While the figures for the bounded-width dataset do not conclusively show the exponential behaviour of the exact algorithm, the Subset Sum CPU times shown in 5.4 clearly show the exponential growth for the exact algorithm for $n > 20$. Below that non-exponential parts of the algorithm dominate. The approximation algorithm does not exhibit this exponential growth. The Subset Sum instances most clearly show the differences between the exact and approximation algorithms as the exact algorithm has to keep track of all 2^n subsets of jobs as long as no two subsets have identical total time. The approximation algorithm can limit this number after each next job to $O(\frac{n}{\varepsilon})$ subsets, so it only grows polynomially.

The Subset Sum instances suggest that the exact algorithm's worst cases lie in problems where the value of a job is strongly correlated with the time it takes to process it in a given

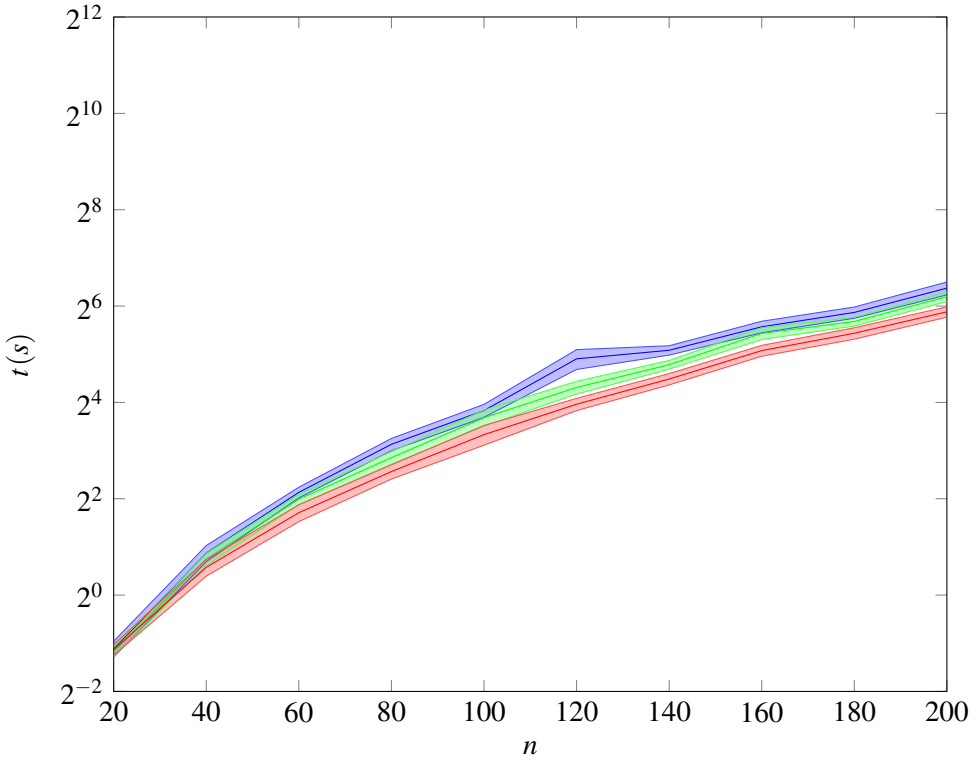


Figure 5.3: CPU time versus number of jobs n for the approximation algorithm with $\varepsilon = 0.1$ on the bounded width dataset. Results for $R = 0.1, 0.5, 0.9$ are shown in blue, green and red respectively. This dataset has 10 instances for each (N, R), a line shows the mean time and an area around it up to one standard deviation away is shaded.

schedule. Figure 5.5 confirms this, as the CPU time of the exact algorithm grows rapidly as the correlation between processing time and revenue nears 1 and setup times near zero. Both algorithms perform about equally well for lower correlation, which explains the nearly equal result for the OAS-SMS dataset, which while slightly different in its construction would lie firmly on the left hand side of the graph. There may be many other problem classes where the exact algorithm exhibits its worst case runtime, but a likely condition for those is that there is not too much of a difference between a job's processing time and setup times on the one hand, and its value on the other.

-

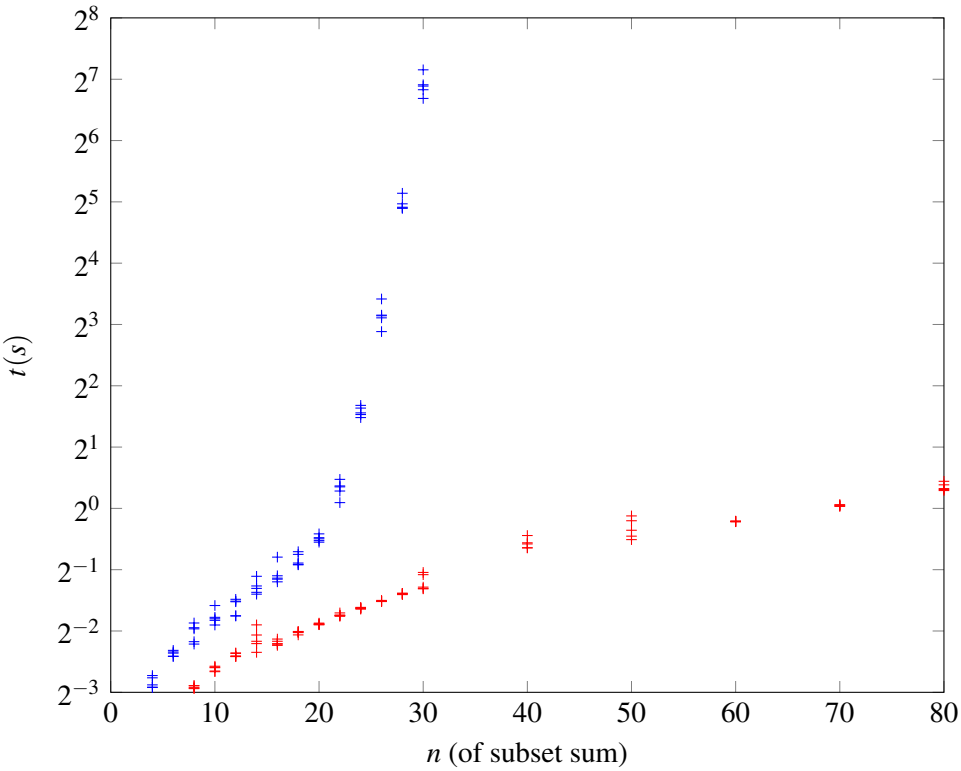


Figure 5.4: CPU time versus problem size n for the exact and approximation algorithms on a collection of Subset Sum problems encoded in bounded-width OAS-SMS instances as knapsack is in the reduction proof in chapter 3. The exact and approximation algorithm CPU times are shown in blue and red respectively.

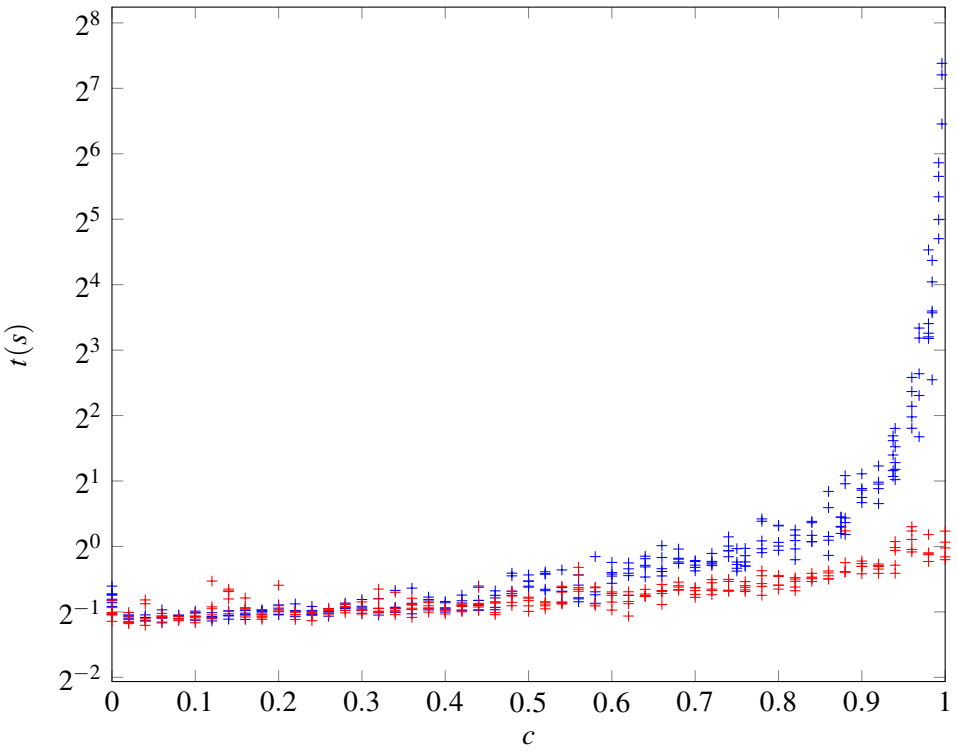


Figure 5.5: Effect of different correlation between job revenue on the one hand and processing times and setup times on the other. Each instance has $n = 30$ and $w = 6$. In blue the CPU time of running the exact algorithm, in red the approximation algorithm.

Chapter 6

Conclusion

In this thesis we investigated the applicability of decision diagrams to order acceptance and scheduling problems as a method of providing bounds on the optimal solution or even finding it, and in particular whether decision diagrams can be adapted to problems where not all jobs are scheduled.

To this end, a special case of the OAS-SMS problem [22] was defined where the maximum number of jobs available at any time point is bounded by a constant, the width. We chose to investigate this problem specifically because it is a generalisation of the travelling salesman problem. In TSP various neighbourhoods, limited by something similar to width [1, 26, 3], are used for local search [11, 12], and the bounded-width used here allowed a more sharp focus on the adaptation of decision diagrams to the order acceptance aspect of OAS problems.

As one of the main advantages of decision diagrams over dynamic programming is its intuitive method for relaxation and restriction, it was necessary to establish the hardness of the problem. Through a reduction of the 0/1 knapsack optimisation problem and the development of a pseudopolynomial algorithm we established that the bounded-width OAS-SMS problem is weakly NP-hard.

An exact decision diagram formulation was developed in chapter 3. Because feasible solutions to an order acceptance and scheduling problem can consist of a variable number of jobs placed in some order, this decision diagram does not have layers associated with each decision variable. Instead, top-down compilation is done in the order of a *perspective*, a way to group and order all possible vertex states in the decision diagram such that all edges go from earlier groups to later groups. Various domination rules were used to stop further exploration of dominated vertices.

On a slight tangent, we showed that the Balas-Simonetti neighbourhood [3], which is still in use [12], in a travelling salesman problem is a subclass of a bounded-width TSP, and that the exact algorithm we developed solves this bounded-width TSP neighbourhood in the same worst case time bound.

Through a merge rule a restricted and relaxed decision diagram formulation was developed which forms a FPTAS for the bounded-width OAS-SMS problem. These merge rules operate in the *perspectives* which allows them to merge partial solutions containing different numbers of jobs, something which would not have worked in a decision diagram

that partitions vertices in layers.

The exact results are shown to outperform state of the art exact algorithms [24] on problems of suitable width in an existing OAS-SMS dataset. Newly generated datasets were used to show the difference in performance between the exact and restricted decision diagrams in hard cases.

6.1 Future work

The algorithms presented here were developed for the bounded-width OAS-SMS problem, but are likely applicable to larger classes of problems. Specifically, one could likely generalise the sequence-dependent setup times to time-dependent setup times, as long as a first in, first out order is maintained so that an earlier completion time of a job always means an earlier start time of its successor. This guarantees that the domains of the decision variables remain discrete, a requirement for decision diagrams. More complicated setup functions could result in a continuous range of possible arrival times.

Similarly, the only condition on the reward function used in proofs of the algorithm is that it is a non-increasing function in completion time, so it too could likely be generalised. Increasing reward functions would lead to a continuous range of completion time and accrued value values, which would break the assumption that solutions can be represented by a sequence of jobs that are each scheduled as early as possible after one another. The case where this assumption breaks is also interesting, but might require large modifications to the decision diagram. Instead of representing discrete states, vertices would have to represent the pieces of a piecewise function.

Another direction of future research could look into broadening the applicability of these algorithms within the OAS-SMS class of problems by softening the width restriction. This is discussed further below in the context of a branch and bound application.

A natural extension of the algorithms could be made for a problem class where every job can be picked an infinite number of times. This would remove the width restriction, as the algorithms are only exponential in jobs for which they have to track whether they have been visited. The FPTAS would however not translate trivially to this new problem, as its approximation guarantee depends on schedules containing at most n jobs.

6.1.1 Satellite Observation Scheduling

For the satellite observation scheduling problem, two extensions would broaden the applicability of the algorithms presented in this thesis. The previously mentioned time dependent setup times more closely model the transition between different jobs as the required movement between two images depends on the angles each of them are taken under, which in turn depend on the time they are taken at. Considering that setup times in this thesis are only used when calculating the start time of a next job when the completion time of the previous job is known, the algorithms proposed here may naturally extend to such setup times.

A more difficult extension which would enhance the applicability to satellite scheduling is to extend it to allow for multiple orbits. Many satellites pass over the same area multiple times in the period during which a job is available and can take a picture in any of them.

In an OAS model of this problem, this could be modelled as multiple machines where each has setup times. Extending the algorithms in this thesis to multiple orbits is not trivial, as taking multiple orbits optimally is not the same as repeatedly taking the optimal single orbit for the remaining jobs.

One approach to incorporate multiple orbits that could be promising is to do the same top-down compilation of the decision diagram through the time windows, but keep track of multiple latest jobs and multiple associated completion times. The visited job state can then be shared between machines (orbits) as schedules are constructed in parallel, unlike in an approach where each orbit is scheduled in order. This would still result in $O(w^k)$ combinations of latest jobs on k machines due to the setup times. There is also a restriction on satellite scheduling which might make it an easier problem. The setup times in the satellite scheduling problem obey a formula based on the change in angle, which is less general than the arbitrary setup times in the OAS-SMS problem. This might be exploited to restrict the possible states of vertices in the decision diagram.

6.1.2 Generalising width

The width used in this thesis for a special case of the OAS-SMS problem uses the availability windows of each job on a timeline, but in essence the width can be seen as *the information about the preceding sequence required to determine the feasibility of potential future sequences*. On a timeline, these are the jobs that could have been done already in the past but also could still be done in the future, and this leads to a sequence of sets of jobs for which the visited state needs remembering.

These sets of jobs do not need to lie in a sequence for this to work. A tree or even a directed graph could work just as well. Such a tree of connected related sets suggests that the width used here can be generalised as treewidth. Not treewidth of the problem graph itself but of the graph of jobs whose order could be either way. Further research could look into how far the width can be generalised while still, for example, forming a neighbourhood that can be explored in polynomial time in a travelling salesman problem. More flexible or larger neighbourhoods can be of use in local search or in branch and bound algorithms.

6.1.3 Usage in a branch and bound algorithm

The relaxed and restricted decision diagrams presented in this thesis provide upper and lower bounds respectively on the optimal solutions of problems, so use in a branch and bound approach seems a natural application of them. This may help to deal with the more difficult dataset in section 4.2 which has difficult instances of limited width.

The merge rule proposed in this thesis only merges states that are close in completion time but identical in their last job and, importantly, visited job state. This means that for the wider OAS-SMS problem the width would still be prohibitive, as was borne out in the results for the approximation scheme on the wider instances in the OAS-SMS dataset. The merge rule does not merge vertices with different visited job states, so these may still grow exponentially. While one may branch on availability windows until the width is reduced far enough before getting bounds, the bounds thus acquired can likely not be translated into a

more global understanding of how the availability windows may be shrunk without losing the optimal solution. In short, in this approach the time saved by branching before getting bounds would come back in the form of more branches to calculate bounds for, with no obvious way based on the work here to use information from one branch to not have to calculate bounds for another.

For a branch and bound approach to the OAS-SMS problem, a merge rule needs to be created which can merge vertices in the decision diagram that have different visited job states. The simplest approach would be to use the union for a restricted diagram and the intersection for a relaxed diagram. Clearly these approaches fulfil the requirements on these diagrams, but they restrict and relax too much, as can be understood from the following example.

Consider an OAS-SMS problem of n jobs which during some extended time window are all available. Clearly in an exact algorithm the number of visited job states could grow to $O(2^n)$. We might choose in a restricted or relaxed decision diagram to limit the number of different visited job states to some number that is not exponential in n, say n^c for some constant c. Now during top-down compilation we would merge these states using their union or intersection. Given that there are about $O(n^c)$ ways to pick c out of n jobs, the resulting vertices after union or intersection would likely have around $n - c$ or c jobs in their visited state, if vertices are merged in similarly sized groups. Any approach that keeps a few unmerged might as well be a greedy algorithm as its success would depend entirely on selecting those few. A better way of merging vertices is needed.

A large part of the difficulty in finding a good merge rule lies in the setup times. Each job has n setup times to it and the same amount from it, so in general it cannot be assumed that any of the n jobs are remotely similar in their setup times. This dissimilarity makes it difficult to use a merge rule that keeps track of the count of jobs that have been merged, but this could still be a promising approach.

Appendix A

Vertex merger for relaxed DDs without layers

In this appendix we give sufficient conditions for which a vertex merger approach results in a valid relaxed decision diagram analogous to those in a recent paper by Hooker [13], but for a generalisation of layers. Below we attempt to follow as much as possible the same layout of the argument as in that paper to facilitate side by side comparison.

A decision diagram D' is a relaxation of decision diagram D if it contains every solution in D with lower or higher cost for minimisation or maximisation problems respectively, neither being a strict inequality, and possibly more solutions. In the remainder of this appendix we will assume a minimisation problem, but the argument applies similarly for maximisation problems.

This relaxation is defined more formally as follows. For every $r-t$ path in D that represents an unambiguous assignment of values to the decision variables $(x_1,\ldots,x_n)$ from their respective domains there is a $r-t$ path in D' corresponding the the same assignment with a shorter or equal path length.

Previous work on decision diagrams uses layers where each vertex is associated with a layer and each layer associated with a decision variable, or sometimes with some stage [14]. These decision diagrams are directed *acyclic* graphs (DAGs) due to their layers. The layers also provide a maximum to the number of edges for any $r-t$ path in it, as each must go to a later layer. These properties, together with the finite discrete domains of each decision variable, guarantee a finite number of paths and vertices in the exact decision diagrams with layers.

We assume the layerless exact decision diagrams to also be without cycles and have a maximum bound to the number of edges in any $r-t$ path in it. Aside from giving the same vertex merger proof again but without mentioning layers, the conditions on it must also guarantee to preserve these properties lest the relaxation creates cycles or an unbounded number of vertices.

Here we introduce some terminology. For any vertex v in a DAG we can identify 3 mutually exclusive sets of vertices that together contain all vertices but v: those that can preceed v in a path, those that can succeed v and those that cannot do either. We say these are predecessor, successor and parallel vertices respectively.

Associated with every vertex in D is a state S which represents the combinations of remaining decisions available for all undecided decision variables. A state S' *relaxes* state S if all decisions available at state S are available at state S' and the cost of each is equal or less in S'. A vertex merger results in valid relaxations of D if three conditions hold.

The first two conditions for a vertex merger that gives a valid relaxation are identical to those of Hooker [13] and repeated below. The third replaces the implicit condition in that paper that merged states must lie in the same *layer* with an explicit broader condition. The first condition says that if one state relaxes another, the same must be true for their successor states for any decision feasible to the other state.

$$\begin{aligned}&\text{If state } S' \text{ relaxes state } S, \text{ then given any decision } v \text{ that is feasible in } S,\\ &\phi(S', v) \text{ relaxes } \phi(S, v).\end{aligned} \tag{C1}$$

The second condition is a requirement on the merge operator, how it must be broad enough to relax both states.

$$S \oplus T \text{ relaxes both } S \text{ and } T. \tag{C2}$$

In decision diagrams with layers, where all edges go from earlier layers to later layers, the acyclic nature of the diagram is preserved because all edges of a relaxed state still can only go to later layers. In a decision diagram without layers this is not the case, and the relaxation of a state might introduce a cycle. Condition C3 prevents this in the most general way.

$$\text{Diagram } D' \text{ resulting from merging } S \text{ and } T \text{ into } S \oplus T \text{ contains only finite paths.} \tag{C3}$$

This condition does not provide instruction for how to achieve this during top down compilaton. One way to do so, as in this thesis, is to find some ordered partition of all possible states such that all decisions can only lead to a state in a later partition. In the presented decision diagrams, time fulfills this role. Another guarantee not provided by this condition is that the repeated application of the vertex merger will result in a finite diagram.

Theorem 3. *If conditions C1, C2 and C3 are satisfied, the merger of vertices with states S and T in diagram D result in a valid relaxation D'*

Proof. Let D be a decision diagram created by top down compilation and let D' be a diagram created by using vertex merger on vertices S and T in D during compilation, creating $S \oplus T$ in their place and compiling further.

Three things must be proven. Any $r-t$ path in D is represented in D', the cost of any such path must be lower or equal in D' and lastly D' itself must be finite and not contain cycles.

The last condition follows directly from condition C3. The diagram D' contains only finite paths and decisions are taken from finite domains. Furthermore, if paths are finite no cycles can exist.

We distinguish two categories of paths in D, those that pass through S or T and those that do not. Those that do not are unchanged by the merger and as such are identical in D' as in D, fulfilling both requirements on them.

Consider a path $r-t$ in D that does pass through the vertex u with state S. Clearly the path $r-u$ exists in D' as $r-u'$ where u' has the merged state $S \oplus T$, with the same cost and the same decisions as top down construction up to S and T was identical and the merger does not affect the cost of edges to $S \oplus T$ from what they were as in the same edges to S and T.

What remains to be proven is that a path $u'-t'$ in D' is present corresponding to $u-t$ in D with the same decisions and the same or lower cost. By condition C2 $S \oplus T$ relaxes S. We prove the relaxation of the remaining path by induction. As the base case, the cost of $r-u$ equals that of $r'-u'$ and $S \oplus T$ relaxes S.

Consider any two successive vertices p and q in the path $u-t$ with states P and Q. Furthermore, assume there is a path $r'-p'$ in D' that corresponds to the same decisions as the path $r-p$ in D and has lower cost, and state P' of vertex p' relaxes state P. Then by condition C1 the decision in p that leads to q is available in p' and leads to some q' with state Q' which relaxes Q. Since P' relaxes P, the cost of the edge $p'-q'$ must be lower or equal to that of $p-q$. Therefore the path $r'-q'$ in D' corresponds to the same decisions as the path $r-q$ in D with lower cost and Q' relaxes Q. By induction then $r'-t'$ is present in D' and correspond to the same decisions as path $r-t$ in D with lower cost.

By symmetry the same is true for paths through the vertex with state T in D. This means all paths $r-t$ in D have an analogue in D' that represents the same decisions and has lower cost. Therefore, D' is a valid relaxation of D. □

No assumption is made about whether D is exact or itself a relaxation of some other diagram, so by repeated application of the vertex merger a relaxation consisting of multiple mergers can be constructed.

This proof forms a first step in extending properties of decision diagrams to cases where vertices in the decision diagram cannot or should not be partitioned into layers. The third condition preventing cycles is of particular use in a direct extension of this proof to restricted decision diagrams or relaxed maximisation problems, and can be constructed similarly but is not given here.

Bibliography

[1] F. Afrati, S. Cosmadakis, C. H. Papadimitriou, G. Papageorgiou, and N. Papakostantinou. The complexity of the travelling repairman problem. *RAIRO-Theoretical Informatics and Applications*, 20(1):79–87, 1986.

[2] S. B. Akers. Binary decision diagrams. *IEEE Transactions on computers*, 6:509–516, 1978.

[3] E. Balas. New classes of efficiently solvable generalized traveling salesman problems. *Annals of Operations Research*, 86:529–558, 1999.

[4] E. Balas and N. Simonetti. Linear time dynamic-programming algorithms for new classes of restricted tsps: A computational study. *INFORMS journal on Computing*, 13(1):56–75, 2001.

[5] D. Bergman, A. A. Cire, W.-J. Van Hoeve, and J. Hooker. *Decision diagrams for optimization*. Springer, 2016.

[6] D. Bergman, A. A. Cire, W.-J. van Hoeve, and J. N. Hooker. Discrete optimization with decision diagrams. *INFORMS Journal on Computing*, 28(1):47–66, 2016.

[7] R. E. Burkard, V. G. Deineko, R. van Dal, J. A. van der Veen, and G. J. Woeginger. Well-solvable special cases of the traveling salesman problem: a survey. *SIAM review*, 40(3):496–546, 1998.

[8] B. Cesaret, C. Oğuz, and F. S. Salman. A tabu search algorithm for order acceptance and scheduling. *Computers & Operations Research*, 39(6):1197–1205, 2012.

[9] A. A. Cire and W. J. van Hoeve. MDD propagation for disjunctive scheduling. In *ICAPS*, 2012.

[10] A. A. Cire and W.-J. van Hoeve. Multivalued decision diagrams for sequencing problems. *Operations Research*, 61(6):1411–1428, 2013.

[11] G. Gutin and A. P. Punnen. *The traveling salesman problem and its variations*, volume 12. Springer Science & Business Media, 2006.

[12] T. Hintsch and S. Irnich. Large multiple neighborhood search for the clustered vehicle-routing problem. *European Journal of Operational Research*, 270(1):118–131, 2018.

[13] J. Hooker. Job sequencing bounds from decision diagrams. In *International Conference on Principles and Practice of Constraint Programming*, pages 565–578. Springer, 2017.

[14] J. N. Hooker. Decision diagrams and dynamic programming. In *International Conference on AI and OR Techniques in Constriant Programming for Combinatorial Optimization Problems*, pages 94–110. Springer, 2013.

[15] T. Ibaraki. The power of dominance relations in branch-and-bound algorithms. *Journal of the ACM (JACM)*, 24(2):264–279, 1977.

[16] E. L. Lawler and J. M. Moore. A functional equation and its application to resource allocation and sequencing problems. *Management Science*, 16(1):77–84, 1969.

[17] M. Lemaıtre, G. Verfaillie, F. Jouhaud, J.-M. Lachiver, and N. Bataille. Selecting and scheduling observations of agile satellites. *Aerospace Science and Technology*, 6(5): 367–381, 2002.

[18] L. Lu, T. E. Cheng, J. Yuan, and L. Zhang. Bounded single-machine parallel-batch scheduling with release dates and rejection. *Computers & operations research*, 36 (10):2748–2751, 2009.

[19] J. Maschler and G. R. Raidl. Multivalued decision diagrams for a prize-collecting sequencing problem. Technical report, Tech. rep., TU Wien, Austria, 2018.

[20] S. Nguyen. A learning and optimizing system for order acceptance and scheduling. *The International Journal of Advanced Manufacturing Technology*, 86(5-8):2021–2036, 2016.

[21] S. Nguyen, M. Zhang, and K. C. Tan. A dispatching rule based genetic algorithm for order acceptance and scheduling. In *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*, pages 433–440. ACM, 2015.

[22] C. Og, F. S. Salman, Z. B. Yalçın, et al. Order acceptance and scheduling decisions in make-to-order systems. *International Journal of Production Economics*, 125(1): 200–211, 2010.

[23] S. Sengupta. Algorithms and approximation schemes for minimum lateness/tardiness scheduling with rejection. In *Workshop on Algorithms and Data Structures*, pages 79–90. Springer, 2003.

[24] Y. L. T. Silva, A. Subramanian, and A. A. Pessoa. Exact and heuristic algorithms for order acceptance and scheduling with sequence-dependent setup times. *Computers & Operations Research*, 90:142–160, 2018.

[25] S. A. Slotnick. Order acceptance and scheduling: A taxonomy and review. *European Journal of Operational Research*, 212(1):1–11, 2011.

[26] J. N. Tsitsiklis. Special cases of traveling salesman and repairman problems with time windows. *Networks*, 22(3):263–282, 1992.