Relaxations of the 3-partition problem

Sebastiaan J.C. Joosten

Department of Applied Mathematics, University of Twente,
Enschede, The Netherlands

Institute for Computing and Information Sciences, Radboud
University Nijmegen, Nijmegen, The Netherland

December 21, 2011

Preface

In order to obtain a master's degree, all I have to do is answer the last question at my final presentation. Before I will be allowed to answer it, however, a number of tasks have to be performed. One of them is writing this thesis.

In order to be able to write this thesis, I needed something to write about. I have been in the fortunate position to be able to do research on a quite wide variety of subjects. Most of this research was carried out at the computer science department of the Radboud University Nijmegen. I was situated in the formal methods group, a very inspiring environment.

Affiliated with this group - and a reason I chose this group - is Hans Zantema. Hans Zantema is a professor at Eindhoven University of Technology, and part time so at Radboud University Nijmegen. As such, Hans Zantema was my supervisor during the course of my master's research.

Hans Zantema proved to be a great supervisor for me. Hans was able to inspire me into investigating a great variety of problems. While discussing my progress, Hans would find new angles, new questions or solutions. By gently steering me in the right direction, Hans was a tremendous help to keep me focused.

My supervisor at my own university, the University of Twente, was Walter Kern. Walter has also been a great help in the research I did. Like Hans, Walter was able to come up with creative angles to look at problems. An important contribution of Walter was to introduce me to bin packing games. This turned out to be a very fruitful research direction, which ultimately led to the main part of this thesis.

For writing this thesis, I decided to make a selection of results obtained during my research. In making this decision, I selected a central topic that could serve as a backbone for related results. This central topic is the 3-partition problem. In particular, the nearly-feasible instances, a type of infeasible 3-partition problem for which a certain half-integral relaxation has a solution. Before I started writing this thesis, I have written an article on this very topic [9]. I wrote this together with my supervisor Hans Zantema, and it is currently in submission. For me, writing this article has been a great practice. In addition, it allowed me to obtain some focus while doing this research.

There are, however, some major differences between the article [9] and this thesis. While the article is focused on the nearly-feasible instances only, this thesis covers other subjects as well. In writing this thesis, I have tried to elaborate on most of the text of the article. Only on a few places is the text in this thesis identical to the text in the article. In those places, I have copied only text that was written by myself.

I hope you enjoy reading this thesis. May it inspire you to ask many ques-

tions. Should you be fortunate enough to have read this thesis before my final presentation, do not hesitate to ask your question there.

Contents

Chapter 1

Introduction

The 3-partition problem is a problem where one has to partition $3q$ numbers (allowing duplicates) into q groups of 3, such that each group has the same sum. If all $3q$ numbers sum to N, this means that every group should have a sum of $\beta = N/q$. The 3-partition problem is a well known NP-complete problem.

One approach to the 3-partition problem, is to compute a set of candidate sets $\mathcal{C}$, which constitutes of those sets that have 3 numbers that sum to β. Let $\mathcal{C}_a$ denote all candidate sets in which the number a occurs. The 3-partition problem only has a solution if the following linear program[1] has an integer solution:

$$\sum_{C \in \mathcal{C}_a} y_C = 1 \quad \text{for all numbers } a$$

A solution to a linear program can be found in polynomial time, while the 3-partition problem is NP-complete. So assuming $P \neq NP$, there should exist instances of 3-partition such that the above linear program is feasible but has no integer solution. In this thesis we will find such instances.

The outline of this thesis is as follows. In Chapter 2 we describe the 3-partition problem, some of its variations, and prove that it is NP-complete. We conclude the chapter, in Section 2.3, by describing the closely related bin-packing games. In Chapter 3 we give instances for which the linear program has $y_C \in \{0, 1/2, 1\}$, but no integer solution. Such instances are called nearly-feasible. We look at different relaxations in Chapter 4. This consists of instances where $y_C \in \{0, 1/3, 2/3, 1\}$, but also instances for some other problems. The chapter concludes with some notes on the bin packing problem.

In order to understand a certain chapter, not all preceding sections need to be read. The dependency graph of the different sections is given in Figure 1.1. The graph is read as follows: An arrow from Section x to Section y means that Section x needs to be understood before reading Section y. This means that Section 2.3 needs to be read before Section 4.3, but Section 4.2 can be skipped.

[1] One might argue that a linear program should always have a linear function that is to be minimized (or maximized). In this thesis, we are only interested in feasibility, so we omit this function.

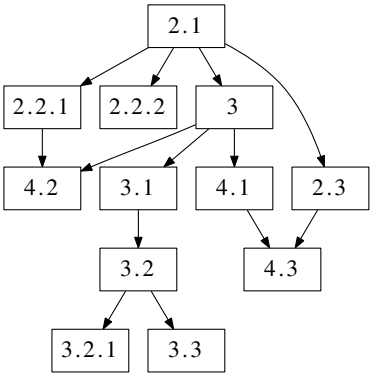


Figure 1.1: Dependencies between sections.

Chapter 2

3-partition problem

This chapter is a general introduction to the 3-partition problem. We begin in Section 2.1 by describing the 3-partition problem and giving a few examples of instances. Section 2.2 gives some background on how the 3-partition problem can be solved. Section 2.3 is about bin-packing games.

2.1 Introduction and notations

A 3-partition problem instance consists of $3q$ numbers, that should be divided into q groups of 3 all having the same sum. Take for example the following problem instances:

- $1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 10', 11$
- $2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 20', 22$
- $0, 1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 20$

For disambiguation, we write a $'$ symbol do distinguish between equal numbers.

In these examples, only the first instance has a solution, namely $\{1, 7, 11\}$, $\{2, 8, 9\}$, $\{3, 6, 10\}$, $\{4, 5, 10'\}$. Note that the sum of each group is 19, and every number is used exactly once. All 12 numbers in the second instance sum to 156. This means that they should be split into four groups with a sum of 39 each (since $4 \times 39 = 156$). Since there are no odd numbers, this is not possible. In the third case, the sum per group should be 30 (again: $4 \times 30 = 120$, which is the total sum). The number 5 in it cannot be combined with two others in such a way that it forms a group of sum 30 (the reader may verify this). We introduce some definitions in order to make observations such as these in a more systematic way.

The 3-partition problem is an example of a decision problem. A decision problem is described by an instance, and a notion of feasibility. A decision problem can be seen as a yes or no question, where "yes" means that the problem is feasible, and "no" means it is not. In this thesis, we mean by feasible that the problem has a solution.

Definition (Decision Problem)**.** *A* decision problem *is described by:*

1. *A set of problem instances D.*

2. A set of feasible problem instances, $Y \subseteq D$.

When describing a decision problem, we usually describe what a problem instance (member of D) will look like, and give a criterion to decide whether the instance is feasible (that is, whether it is in Y or not). As was the case with 3-partition, this criterion can usually be defined in terms of a predicate p:

$$Y = \{i \in D \mid \exists s.\ p(s, i)\}$$

In this case, we call s a *solution* to our problem, and p will be our verification procedure. If p can be computed in polynomial time on a Turing machine, we say that the problem is in the class *NP*.

We give two more definitions in order to define 3PART as a decision problem.

Definition (Power set)**.** *The set of all subsets of some set A is denoted by $\mathcal{P}(A)$, and is called the* power set *of A. The set of all subsets of size k of some set A is denoted by $\mathcal{P}_k(A) = \{S \in \mathcal{P}(A) \mid |S| = k\}$.*

Definition (Partition)**.** *We say that M is a partition of some set S if:*

- *M is a collection of non-empty subsets of S: $M \subseteq \mathcal{P}(S)$*
- *such that no element occurs in two of these subsets: $\forall S_1, S_2 \in M.\ S_1 = S_2 \vee\ S_1 \cap S_2 = \emptyset$.*
- *and every element occurs once: $\bigcup M = S$.*

Definition (3-Partition problem)**.** *The 3-partition problem (*3PART*) is a decision problem given by (A, w) having $|A| = 3q$ and $w : A \to \mathbb{N}$. The objective is to find $\beta \in \mathbb{N}$ and a $M \subseteq \{S \in \mathcal{P}_3(A) \mid \sum_{a \in S} w(a) = \beta\}$, such that M is a partition of A. The feasible problem instances are those instances for which such M exists.*

For giving an instance, we write $w(a_1), \ldots, w(a_{3q})$ for $a_1, \ldots, a_{3q} \in A$. For any set S, the notation $w(S)$ will be used as an abbreviation for $\sum_{a \in S} w(a)$. Since we know q and the total sum of all sets, we also know the sum per set. We refer to this sum per set $\beta = w(A)/q$.

A set of three elements $\{a, a', a''\} \subseteq A$ such that $w(a) + w(a') + w(a'') = \beta$ is called a *candidate set*. The set of all candidate sets is denoted by $\mathcal{C}$:

$$\mathcal{C} = \{S \in \mathcal{P}_3(A) \mid w(S) = \beta\} \tag{2.1}$$

Note that $|\mathcal{C}| \leq \binom{k}{3} \in O(k^3)$, so calculating $\mathcal{C}$ is polynomial.

A solution of 3-PART consists of a vector $\vec{y} \in \{0, 1\}^{\mathcal{C}}$ such that:

$$\begin{aligned} \sum_{C \ni e} \vec{y}_C = 1 \quad &\text{for all } e \in A, \\ \vec{y} \in \{0, 1\}^{\mathcal{C}}&. \end{aligned} \tag{2.2}$$

Note that the formulation above is an integer linear program, for which general solvers exist. We will come back to this in Section 2.2.2.

Once we have obtained this system of equations (2.2), we have easy arguments to why the second and third example have no solution: For the second instance, there are no candidate sets. Therefore all equations read $0 = 1$, which

shows a contradiction. In the third instance, there are four candidate sets. The equation about the element 5 reads $0 = 1$.

The equations also help us to find a solution for our first instance. The reader may verify that, by choosing $\vec{y}_{\{2,7,10'\}} = 1$, the values of $\vec{y}$ are determined by the equations and we obtain the solution $\{2, 7, 10'\}$, $\{1, 8, 10\}$, $\{3, 5, 11\}$, $\{4, 6, 9\}$.

2.2 Solving 3PART

It might seem like 3PART is a rather artificial problem: there are not many practical situations in which one has to divide $3q$ integers in q triples of equal sum. Nevertheless, 3PART is an important problem in combinatorics because it belongs to one of the many NP-complete problems. A proof of this is shown in [6]. NP-completeness implies that there is a certain difficulty to solving the problem. We will learn about this in Section 2.2.1. Nevertheless, NP-complete problems can be solved. We will look at various ways we might do this, using known algorithms, in Section 2.2.2.

2.2.1 NP-completeness

Definition (NP-complete)**.** *A decision problem is* NP-complete *if:*

1. *P is NP. This means that there is a Turing machine that, given a feasible decision problem and a proof, will verify (if the proof is of the right form) that the decision problem is feasible in polynomial time (with respect to the problem size, not to the proof size).*

2. *For any problem P' in NP, there is a polynomial-time reduction from P' to P.*

The first known NP-complete problem is the *satisfiability problem*.

Definition (SAT)**.** *An instance of the satisfiability problem (*SAT*) is stated as a sentence using predicates $p_1, \ldots, p_n$ and the operators "or" $\vee$ and "negation" $\neg$. The feasible instances are those for which one can assign "true" T or "false" F to each predicate such that the sentence holds.*

That is: let the assigned predicates $p_1, \ldots, p_n$ reduce to whatever is assigned to them, let $T \vee T$, $T \vee F$, $F \vee T$ and $\neg F$ reduce to T, and let $F \vee F$ and $\neg T$ reduce to F. If the sentence reduces to T, we say that it holds.

It can be shown that the class NP consists of those problems that can be reduced in polynomial time (with respect to the input size of the problem) to a satisfiability problem. By *reduced* (or a *reduction*) to problem X, we mean reformulated in such a way that X has a solution if and only if the original problem does. This result is known as the *Cook-Levin theorem*, which was independently proven by Thomas Cook [1] and Leonid Levin [14]. We do not get into the technical details here. Trivially, the satisfiability problem can be reduced to itself in polynomial time (by not changing anything), so it is in NP itself. It is therefore called NP-complete.

To prove that any other problem is NP-complete, we can show that there is a polynomial-time reduction from the satisfiability problem, and apply the Cook-Levin theorem. There are many NP-complete problems, of which some 300 can

be found in [7]. This reference uses 3-PART as a "basic" NP-complete problem, from which many other NP-completeness results are proven. It also contains a proof of the NP-completeness of 3-PART. This is done via a reduction from 3-dimensional matching, via the 4-partition problem. We will give a modified, slightly stronger version of this proof, which shows NP-completeness of the numerical 3-dimensional matching.

Definition (NkDM)**.** *The Numerical k-dimensional matching (*NkDM*) consists of k sets $S_1, \ldots, S_k$ containing q distinct elements each, a weight function for each set: $w_i : S_i \to \mathbb{N}$, and a bound β. Candidate sets $\mathcal{C}$ are given by:*

$$\mathcal{C} = \{\{s_1, \ldots, s_k\} \mid (s_1, \ldots, s_k) \in S_1 \times \cdots \times S_k,\ w_1(s_1) + \cdots + w_k(s_k) = \beta\}$$

The objective is to find a set $M \subseteq \mathcal{C}$ such that M is a partition of $S_1 \cup \ldots \cup S_k$.

Definition (3-dimensional matching)**.** *A* 3-dimensional matching *consists of a set $M \subseteq X \times Y \times Z$ having $|X| = |Y| = |Z|$ such that X, Y and Z are disjoint. It is solvable if there is an $M' \subseteq M$ such that $|M'| = |X|$ and $\bigcup M' = X \cup Y \cup Z$.*

It was established by Karp [10] that the 3-dimensional matching problem is NP-complete.

Theorem 1. *N4DM is NP-complete*

Proof. This proof is based on [7]. It is easy to see that N4DM is in NP. To show it is NP-hard, we give a reduction from 3-dimensional matching.

Let $X = \{x_1, \ldots, x_q\}$, $Y = \{y_1, \ldots, y_q\}$ and $Z = \{z_1, \ldots, z_q\}$ be the sets in some 3-dimensional matching problem given by M. Without loss of generality (if not, we would have a polynomial-time argument why this instance is not feasible), assume that $\bigcup M = X \cup Y \cup Z$.

We create a N4DM with sets $S_1, \ldots, S_4$. For each $(x_i, y_j, z_k) \in M$, there is an element in each of these sets (so $|S_n| = |M|$). Let $s_1, \ldots, s_4$ be the corresponding elements, then the weights are:

$$w_1(s_1) = \begin{cases} 2q^3 + i \cdot q^2 & \text{once} \\ i \cdot q^2 & \text{otherwise} \end{cases}$$

$$w_2(s_2) = \begin{cases} j \cdot q & \text{once} \\ q^3 + j \cdot q & \text{otherwise} \end{cases}$$

$$w_3(s_3) = \begin{cases} k & \text{once} \\ q^3 + k & \text{otherwise} \end{cases}$$

$$w_4(s_4) = 2q^3 - i \cdot q^2 - j \cdot q - k$$

The bound β is $4q^3$. By "once", we mean that for each i (j, k), there is exactly one element s_1 (s_2, s_3) that gets the corresponding value. Such elements are called "actual" elements, the others are called "dummy" elements. Note that every candidate set $\{s_1, s_2, s_3, s_4\}$ contains either three actual elements, or three dummy elements, and that all weights of s_4 are different.

A solution of N4DM can be obtained from a 3-dimensional matching solution M' by taking the actual elements corresponding to $(x_i, y_j, z_k) \in M'$, and using the dummy elements for $(x_i, y_j, z_k) \notin M'$. Vice versa, a solution to a

N4DM yields one for the original 3-dimensional matching by taking those M that correspond to an s_4 that are linked to actual elements only. Therefore the N4DM is NP-hard. □

Theorem 2. *N3DM is NP-complete*

Proof. This proof is loosely based on [7]. It is easy to see that N3DM is in NP. To show it is NP-hard, we give a reduction from N4DM.

Let $S_1, \ldots, S_4$ with weight functions $w_1, \ldots, w_4$ be the original N4DM instance having target sum β and $q = |S_1|$. We create a N3DM with the sets T_1, T_2, T_3 having $q + q^2$ elements each, weights w'_1, w'_2, w'_3 and as bound 7β.

"Regular" elements correspond to the original elements in $S_1, \ldots, S_4$. The regular elements from S_1 and S_2 go in T_1 and T_2, respectively. The regular elements from S_3 and S_4 go in T_2 and T_3. In addition:

- T_1 contains "paring" elements $P = S_3 \times S_4$. $T_1 = S_1 \cup P$
- T_2 contains "filler" elements F, $|F| = q^2 - q$ in total. $T_2 = S_2 \cup S_3 \cup F$
- T_3 contains "co-parinig" elements $P' = \{\text{co}(p) \mid p \in P\}$ (one for every pairing element). $T_3 = S_4 \cup P'$

The reader should verify that $|T_1| = |T_2| = |T_3| = q^2 + q$. All weights in the original problem are less than β, so we add β a couple of times to distinguish elements of different types. This is done such that every candidate set $\{t_1, t_2, t_3\}$ either contains:

- An element from S_1, one from S_2, and one co-pairing element
- An element from S_3, one from S_4, and one pairing element
- A filler element, a pairing and a co-pairing element corresponding to the pairing element.

$$
\begin{array}{ll}
\forall s_1 \in S_1 & w'_1(s_1) = w_1(s_1) \\
\forall s_2 \in S_2 & w'_2(s_2) = w_2(s_2) + 4\beta \\
\forall s_3 \in S_3 & w'_2(s_3) = w_3(s_3) + 2\beta \\
\forall s_4 \in S_4 & w'_3(s_4) = w_4(s_4) \\
\forall s_3 \in S_3, s_4 \in S_4 & w'_1(\{s_3, s_4\}) = 7\beta - w'_2(s_3) - w'_3(s_4) \\
 & \qquad = 3\beta - w_3(s_3) - w_4(s_4) \\
\forall s_3 \in S_3, s_4 \in S_4 & w'_3(\text{co}(\{s_3, s_4\})) = 7\beta - w'_1(\{s_3, s_4\}) \\
 & \qquad = 4\beta + w_3(s_3) + w_4(s_4) \\
\forall f \in F & w'_2(f) = 0
\end{array}
$$

If there exists a solution for N4DM, then for every selected candidate set $\{s_1, s_2, s_3, s_4\}$, select $\{s_1, s_2, \text{co}(s_3, s_4)\}$ and $\{(s_3, s_4), s_3, s_4\}$. The remaining $q^2 - q$ pairing elements can be combined with co-pairing and filler elements: $\{p, f, \text{co}(p)\}$.

If the N3DM has a solution, note that there are only $q^2 - q$ filler elements. Hence - to match every pairing element - q of the used candidate sets are of the form $\{p, s_3, s_4\}$, and - to match the co-pairing elements - another q are of the form $\{s_1, s_2, p'\}$. We can combine these pairs such that $w'_1(p) + w'_3(p') = 7b$. This yields the desired $\{s_1, s_2, s_3, s_4\}$ as the solution to N4DM. □

Theorem 3. *3PART is NP-complete*

Proof. We give the straightforward reduction from N3DM. Our set of elements A is simply the union of the sets in the N3DM (or $A = S_1 \cup S_2 \cup S_3$). Weights are changed such that the candidate sets remain the same. If β is the target sum of the N3DM instance, and w_i are the N3DM weight-functions, then let the 3PART weight-function be $w(a) = w_i(a) + 3 \cdot i \cdot \beta$ for $a \in S_i$. The target sum of the 3PART is $19 \cdot \beta$. Solutions of the 3PART instance are solutions of the N3DM and vice-versa. □

2.2.2 Using solvers

For solving 3PART directly, one can either use heuristics [2], or use methods that are slower than $O(n^k)$ for any k. Calculating the candidate sets $\mathcal{C}$ as in equation (2.1), we have transformed the problem into an exact set-cover problem. There are efficient implementations for solving set-cover problems, for example the "Dancing Links" algorithm [12]. In this section, we look at how a SAT-solver would solve the problem, and at how an ILP-solver would. Since implementations of such solvers vary, we don't give complete descriptions or algorithms, but show an example run based on the 3PART problem given by the weights:

$$1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 10', 11$$

To get to (2.2), we define and number the candidate sets:

Number	$\{a$	a'	$a''\}$
1	1	7	11
2	1	8	10
3	1	8	10'
4	2	7	10
5	2	7	10'
6	2	8	9
7	3	5	11
8	3	6	10
9	3	6	10'
10	3	7	9
11	4	5	10
12	4	5	10'
13	4	6	9
14	4	7	8
15	5	6	8

According to this numbering, we are looking for a vector $\vec{y} \in \{0,1\}^{15}$. An ILP solver would try to find a vector y for which (2.2) holds having $\vec{y} \in [0,1]^{15}$. This can be done in polynomial time, but the resulting $\vec{y}$ might not be integer.

Take for example:

$$\vec{y}_1 = \vec{y}_3 = \vec{y}_5 = \vec{y}_6 = \vec{y}_7 = \vec{y}_8 = \vec{y}_{11} = \vec{y}_{13} = 1/2$$
$$\vec{y}_i = 0 \quad \text{all other } i$$

To find an integer solution, the ILP-solver can try two things:

1. Try adding $y_1 = 1$ as a constraint and solve again. If there is a solution, and it is integer, we are done. If there is no solution, we have deduced

$y_1 = 0$. If there is a solution, but it is not integer, do the same trick on the next non-integer variable. This method is called branching. ILP solvers have heuristics for selecting non-integer variables.

2. Add a constraint such as:

$$\vec{y}_8 + \vec{y}_{11} + \vec{y}_{13} \leq 1$$

We know this constraint must hold since the three sets mutually share an element. However, the constraint does not hold in the solution found, so we add it and try to solve our problem again. If no more such constraints can be added, and the problem is still feasible, then there are integer solutions. This step is called adding a cutting plane, since the inequality $\vec{y}_8 + \vec{y}_{11} + \vec{y}_{13} \leq 1$ can be drawn as a hyperplane in the solution-space of $\vec{y}$.

If we wish to solve our problem by a SAT-solver, we should use Boolean variables instead of integer variables. We choose to formulate this as:

$$\begin{aligned} \bigvee_{C \ni e} p_C \quad & \text{for all } e \in A, \\ \neg p_C \vee \neg p_{C'} \quad & \text{for all } C, C' \in \mathcal{C} \text{ for which } C \cap C' \neq \emptyset \end{aligned} \tag{2.3}$$

Note that this formulation is at worst quadratic in $|\mathcal{C}|$.[1] A SAT-solver takes a logical formula as its input. By taking the conjunction of (2.3), we obtain such a logical formula in conjunctive normal form. The formula has a solution if and only if the original 3PART instance is feasible.

Note that the formula is of a very specific normal form: it is the conjunction of clauses with either only positive terms, or exactly two negative terms. Since 3PART is NP-complete, a SAT problem of this specific form is NP-complete as well.

2.3 Bin packing problems

In the previous section, we have seen the 3-dimensional matching and numerical 3-dimensional matching problems. In this section, we look at bin packing problems. As in Section 2.2, we will denote $w(S) = \sum_{a \in S} w(a)$ for any set S.

Definition (Bin packing problem)**.** *A* bin packing problem *is described by a set of elements A with weights $w : A \to \mathbb{N}$, a number of bins q and a bin size β. The candidate sets are those subsets of A which "fit in a bin".*

$$\mathcal{C} = \{S \in \mathcal{P}(A) \mid w(S) \leq \beta\}$$

A solution to the bin packing problem consists of q such candidate sets $M \subseteq \mathcal{C}$, such that M is a partition of A.

[1] There are linear-size (in terms of $|\mathcal{C}|$) transformations to SAT as well. Such transformation require us to use more variables and this would make it more complicated to see that the transformation is sound.

It is common to look at the least number of bins needed for a bin packing problem, and different algorithms to compute this have been developed [15, 16, 5, 8]. Our interest in the bin packing problem, however, originates from cooperative game-theory.

In relation to the bin packing problem, Faigle and Kern [3] defined bin packing games, and the ϵ-core of a game. A cooperative game is defined by a set of players, N, and a value function $v : \mathcal{P}(N) \to \mathbb{R}$ such that $v(\emptyset) = 0$. The ϵ-core of a game is the set of functions $x : N \to \mathbb{R}$ satisfying:

$$\begin{aligned} x(N) &\leq v(N) \\ \forall S \in \mathcal{P}(N) \quad x(S) &\geq (1-\epsilon)v(S) \end{aligned}$$

In a bin packing game the players are given by bins and elements. We give a slightly modified version of the bin packing game in order to be more concise on notations.

Definition (Bin packing game)**.** *Let B be a set of q bins of size $\beta \in \mathbb{N}$, and A a set of elements of some size $w : A \to \mathbb{N}$ such that $0 \leq a \leq \beta$ for $a \in A$ Candidate sets are defined as:*

$$\mathcal{C} = \left\{ S \in \mathcal{P}(A) \;\middle|\; \sum_{e \in S} w(e) \leq \beta \right\}$$

A bin packing game *is a game with players $A \cup B$ and value function v:*

$$v(S) = \max \sum_{j=1}^{|S|} w(I_j)$$

where the maximum is taken over all collections of pairwise disjoint subsets $I_1, \ldots, I_{|S|} \in \mathcal{C}$.

Some research was done on whether the ϵ-core of a bin packing game is empty. In [4] it was shown that for $\epsilon > 0$ the ϵ-core is nonempty if β is large enough ($\beta \geq 48\epsilon^{-5}$ for $\epsilon^{-1} \in \mathbb{N}$). There are bin packing games with an empty ϵ-core if $\epsilon < \frac{1}{7}$ [3]. In [13] it was shown that this value $\frac{1}{7}$ cannot be increased if all weights are $> \beta/3$. In [11] it is shown that for $\epsilon \geq 1/3 - 1/108 = 35/108$, the ϵ-core is nonempty.

If all weights are $> \beta/3$, then the number of items per bin is at most two. In this case, the least ϵ such that the ϵ-core is nonempty is known to be $\frac{1}{7}$. Allowing three items per bin complicates things, and no exact bound for ϵ is known. Even for the case where weights are strictly between $1/2$ and $1/4$, this value is not known.

To compute the least ϵ such that the ϵ-core of a game is nonempty, it suffices to compute $v(N)$ and a solution to the following Linear Program (LP):

$$\begin{aligned} \max_{\vec{x}} \quad & \sum_{S \in \mathcal{C}} w(S) \cdot \vec{x}_S & & \\ \text{subject to} \quad & \sum_{S \ni a} \vec{x}_S & & \text{for all } a \in A \\ & \sum_{S \in \mathcal{C}} \vec{x}_S \leq q & & \\ & 0 \leq \vec{x}_S \leq 1 & & \text{for all } S \in \mathcal{C} \end{aligned} \tag{2.4}$$

Let $\vec{x}^*$ be the solution to this LP, and v' the corresponding value:

$$v' = \sum_{S \in \mathcal{C}} w'(S) \cdot \vec{x}^*_S$$

Then the ϵ-core is nonempty if and only if $\epsilon \geq (v' - v(N))/v'$ [4]. Note that another way to define $v(N)$ for bin packing games is via the following (ILP):

$$\begin{array}{lll} \max\limits_{\vec{x}} & \sum\limits_{S \in \mathcal{C}} w(S) \cdot \vec{x}_S & \\ \text{subject to} & \sum\limits_{S \ni a} \vec{x}_S & \text{for all } a \in A \\ & \sum\limits_{S \in \mathcal{C}} \vec{x}_S \leq q & \\ & \vec{x}_S \in \{0,1\} & \text{for all } S \in \mathcal{C} \end{array} \tag{2.5}$$

If $\vec{x}^*$ is the solution of this ILP, then $v(N) = \sum_{S \in \mathcal{C}} w(S) \cdot \vec{x}^*_S$. Note that there is a strong similarity between this ILP and (2.2). This leads us to consider different relaxations of (2.2).

Chapter 3

Nearly-feasible instances for 3PART

In this chapter, we focus on 3PART. Whenever we refer to instances or solutions, we mean instances and solutions to 3PART. Let $w : A \to \mathbb{N}$ indicate the weights of the elements A of the 3PART instance. We write $w(S) = \sum_{a \in S} w(a)$ for any set S. The set $\mathcal{C}$ indicates the candidate sets for the 3PART instance, which are defined as in (2.1), that is:

$$\mathcal{C} = \{S \in \mathcal{P}_3(A) \mid w(S) = \beta\} \tag{3.1}$$

Analogous to the LP relaxation (2.4), we look at the LP relaxation of (2.2).

$$\begin{aligned} \sum_{C \ni e} \vec{y}_C = 1 \quad \text{for all } e \in A, \\ \vec{y} \in [0,1]^{\mathcal{C}}. \end{aligned} \tag{3.2}$$

Whenever (3.2) has a solution, there exists a fractional solution to (3.2). This means that for some integer M, $\vec{y} \cdot M$ is integer again. In order to distinguish between these different solutions of (3.2), we can multiply the entire formulation by this factor M. This yields (3.3).

$$\begin{aligned} \sum_{C \ni e} \vec{y}_C = M \quad \text{for all } e \in A, \\ \vec{y} \in \{0, \ldots, M\}^{\mathcal{C}}. \end{aligned} \tag{3.3}$$

If we demand that $M = 1$, we obtain the original formulation (2.2). Hence 3PART instances for which (3.3) has an $M = 1$ solution are precisely the feasible 3PART instances. If we demand $M = 2$, we obtain those instances for which (3.2) has a half-integral solution. This includes the feasible problems, plus some other problems. Those other problems are the instances called "nearly-feasible" [9].

Definition (Nearly-feasible)**.** *A* nearly-feasible *3PART instance (A, w) is a 3PART instance for which (3.3) has a solution for $M = 2$, but not for $M = 1$.*

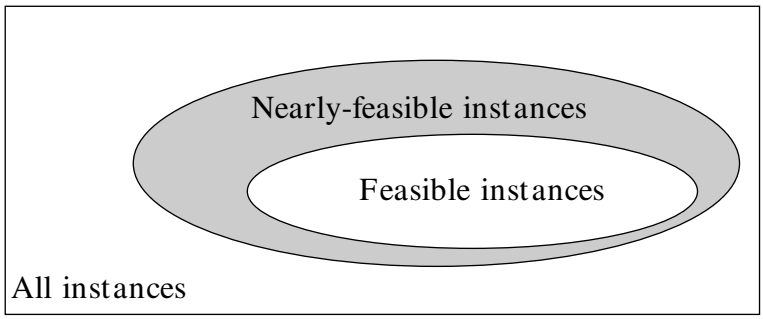


Figure 3.1: Instances and feasibility

As an example of a nearly-feasible 3PART instance, consider the following weights:

$$0, 1, 2, 3, 4, 5, 10, 16, 19, 24, 25, 28, 29, 36, 37, 45, 46, 70, 82, 84, 85, 88, 100, 113$$

There are 17 feasible sets, and (3.3) has a solution with $M = 2$. We look at this solution in table 3.1.

$C = \{a,$	$a',$	$a''\}$	$\vec{y}_C$
0,	19,	100	1
0,	37,	82	1
1,	5,	113	1
1,	36,	82	1
2,	4,	113	1
2,	29,	88	1
3,	16,	100	1
3,	28,	88	1
3,	46,	70	0
4,	45,	70	1
5,	29,	85	1
10,	24,	85	1
10,	25,	84	1
16,	19,	84	1
24,	25,	70	1
28,	45,	46	1
36,	37,	46	1

Table 3.1: A solution for (3.3) where $M = 2$

It is not immediate that 3.3 does not have a solution with $M = 1$. We will illustrate this by looking at cycles in a graph. The same technique can be applied to all the graphs in Figure 3.5, but we only give one graph as an example.

Assume 3.3 has, for our example, a solution for $M = 1$. This means there are 8 candidate sets in $\mathcal{C}$ which together form a partition of A. We have drawn 16 candidate sets in Figure 3.2, with a line between candidate sets whenever they share an element. Hence two candidate sets that share a line cannot occur

together in the partition of A. We have drawn all but one candidate set, so we should be able to find 7 disjoint candidate sets in the figure. This means that we should be able to find 7 independent vertices. We can partition all of the candidate sets drawn in the figure into three cycles of length 3, and one cycle of length 7. If we are to select independent vertices, we can select at most one of the candidate sets from the cycles of length 3 (in total 3), and three from the cycle of length 7 (another 3). Hence we are not able to find 7 independent vertices in the graph, and 3.3 does not have a solution for $M = 1$.

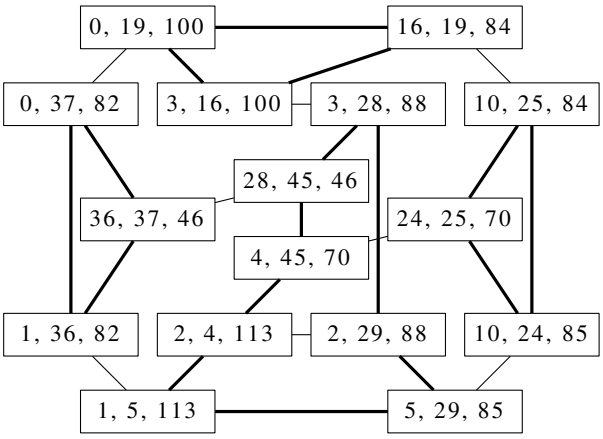


Figure 3.2: Solution graph of the example

3.1 Solution graphs

We formalize the way we draw graphs like the one in Figure 3.2. This will provide us with useful notions about nearly-feasible instances. Note that we based the graph on the candidate sets having $\vec{y}_C = 1$ in an $M = 2$ solution of (3.3). Such a graph is called the solution graph. In case there are two elements shared between two candidate sets, we wish to draw this as two edges. For this reason, we define the solution graph as a multigraph.

Definition (Multigraph)**.** *A* multigraph (V, E) *consists of a non-empty set of vertices V, and a function $E : \mathcal{P}_2(V) \to \mathbb{N}$ indicating the number of edges between two vertices.*

Definition (Solution graph)**.** *Let $\vec{y}$ be a solution of 3.3 for $M = 2$. The* solution graph *of $\vec{y}$ is a multigraph defined by*

$$V = \{C \in \mathcal{C} \mid \vec{y}_C = 1\}\,, \quad E(\{C, C'\}) = |C \cap C'| \tag{3.4}$$

Note that V should be non-empty in order for the solution graph to be properly defined. If $\vec{y}_C \in \{0, 2\}$, this will not be the case. However, $\vec{y}_C \in \{0, 2\}$ implies that (3.3) has a solution for $M = 1$ (dividing everything by 2). For nearly-feasible instances, there is a solution $\vec{y}$ such that V is nonempty and the solution graph is properly defined. For this reason, we restrict to such $\vec{y}$'s, which we shall call a nearly-feasible solution.

Definition (Nearly-feasible solution)**.** *Consider a nearly-feasible 3PART instance, with $\vec{y}$ a solution of 3.3 having $M = 2$. Then $\vec{y}$ is called a* nearly-feasible solution.

We state some properties that hold for every solution graph of a nearly-feasible solution.

Theorem 4. *Let (V, E) be the solution graph of nearly-feasible solution $\vec{y}$. Then (V, E) is* cubic*, that is: $\forall C \in V.\ \sum_{C' \neq C} E(\{C, C'\}) = 3$.*

Proof. Let A be the elements of the original 3-PART instance.

By (3.3), $\sum_{C \ni a} \vec{y}_C = 2$. For $C \in V$ we have $\vec{y}_C = 1$, so for every $a \in C$ there is exactly one $C' \neq C$ having $a \in C'$ and $\vec{y}_C = 1$. So, for $C \in V$: $\sum_{C' \neq C} |C \cap C'| = |C| = 3$. Hence $\forall C \in V.\ \sum_{C' \neq C} E(\{C, C'\}) = 3$. □

Theorem 5. *Let $\mathcal{C}$ be the candidate sets of an infeasible 3PART instance (A, w) with $|A| = 3q$. Any $q - 1$ candidate sets are not pairwise disjoint.*

Proof. Assume there are $q - 1$ such disjoint candidate sets. The total number of elements in the 3PART instance is $3q$, so there are exactly three elements a_1, a_2, a_3 not in these $q - 1$ disjoint candidate sets. The weight of each of the $q - 1$ disjoint candidate sets is β and the total weight of all $3q$ elements is $\beta \cdot q$. We conclude that $w(a_1) + w(a_2) + w(a_3) = \beta$, so $\{a_1, a_2, a_3\}$ is a candidate set. Define $\vec{y}'$ by

$$\vec{y}'_C = \begin{cases} 1 & C \text{ is one of the } q-1 \text{ disjoint candidate sets or } C = \{a_1, a_2, a_3\} \\ 0 & \text{otherwise.} \end{cases}$$

By construction $\vec{y}'$ is a solution to (3.3) for $M = 1$, contradicting the assumption that the instance is nearly-feasible. □

Corollary 1. *In a solution graph (V, E) corresponding to the solution of a nearly-feasible instance (A, w) with $|A| = 3q$, there are no $q - 1$ pairwise independent vertices in V.*

Proof. Immediate by the definition of a solution graph. □

In some cases, we can find a smaller nearly-feasible instance in a straightforward way by looking at the solution graph. In order to exclude such cases, we wish to restrict ourselves to nearly-feasible instances that are smallest in some sense.

Definition (Locally smallest)**.** *A nearly-feasible instance with $|A| = 3q$ is* locally smallest *if for every nearly-feasible solution, its solution graph (V, E) has these properties:*

- *(V, E) is connected*
- *$|V| = 2q$*

The following theorem states that our notion of locally smallest makes sense:

Theorem 6. *Any nearly-feasible instance with $|A| = 3q$ and q minimal, is locally smallest.*

Proof. To see that (V, E) is connected, we use minimality of q. Assume (V, E) is not connected. Let V' be any connected component. Note that $A - \bigcup V'$ with the original weight-function is a 3-PART instance. It must have a solution to (3.3), since we can take the original values in g. This instance has a smaller q (since V' is non-empty), so it cannot be nearly-feasible. Therefore, $A - \bigcup V'$ must have a solution to (2.2). Using the same argument, $\bigcup V'$ must also have a solution to (2.2). By combinding the two solutions, A must have a solution to (2.2) too. contradicting that the original 3-PART instance is nearly-feasible. We conclude that (V, E) is connected. Note that we come to the same contradiction when we let $V' = V$ if $A \subset \bigcup V$. Hence we conclude $A = \bigcup V$.

Every candidate set contains three elements, so $|V| \cdot 3 = \sum_{C \in V} |C|$. As we have seen in the proof of Theorem 4, every element occurs in exactly two candidate sets: $3|V| = 2\,|A|$. By definition of a 3-PART instance, $|A| = 3q$. Hence $|V| = 2q$.

Since (V, E) is connected and $|V| = 2q$, the instance is locally smallest. □

The solution graph drawn in Figure 3.2 is the unique solution graph for our example instance. This means that the instance in our example is locally smallest. If we would have added three elements with weights $0, 0, 119$ to our example, Figure 3.2 would still correspond to a solution graph, but the instance would not be locally smallest.

3.2 Finding nearly-feasible instances using solution graphs

In the previous section, we have seen how we can create solution graphs from nearly-feasible instances. In this section, we will try to create nearly-feasible instances from solution graphs. Our aim in this, is to find a smallest nearly-feasible instance, in terms of q. To do so, we find multigraphs that could be solution graphs of locally smallest nearly-feasible instances. For this reason, we look at connected cubic multigraphs.

The number of connected cubic multigraphs on $2n$ points is, up to isomorphism, $1, 2, 6, 20, 91, 509, 3608, 31856$ for $n = 1, 2, 3, 4, 5, 6, 7, 8$[18]. To generate these multigraphs, we can use the "`genbg`" package from the "`nauty`" program [17]. After generating the multigraphs, we check the number of independent points it has based on Corollary 1.

Every connected cubic multigraph on $2q$ points has an independent set of size $q - 1$ for $q \leq 5$. There are, up to isomorphism, 14 connected cubic multigraphs on 12 points with no independent set of size 5. These graphs are drawn in Figure 3.3. Generating these graphs takes less than a second. Details on how we obtained these graphs are given in Appendix A.2.

The next question we ask ourselves is: which of these multigraphs is a solution graph. A locally smallest nearly-feasible instance to which either of these graphs is a solution graph, will be a nearly-feasible instance for which q is minimal. This follows from the fact that there are no smaller solution graphs.

To ensure that a multigraph is the solution graph of some nearly-feasible instance A with weights w, we need that the instance has a candidate set for each vertex. Also, whenever two candidate sets are joined by one or two edges, there have to be one or two elements they have in common. In other words,

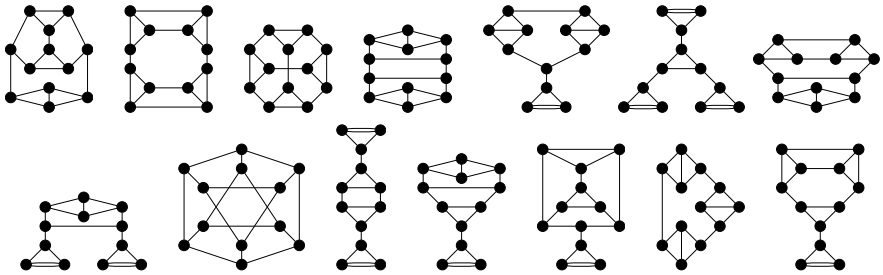


Figure 3.3: All connected cubic multigraphs on 12 points with no independent set of size 5.

we obtain our nearly-feasible instance by taking an element for each edge, and requiring that the candidate sets to which they are incident contain them, and have the correct total sum. The principle is illustrated in Figure 3.4.

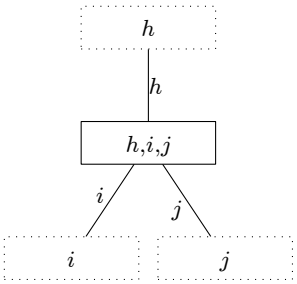
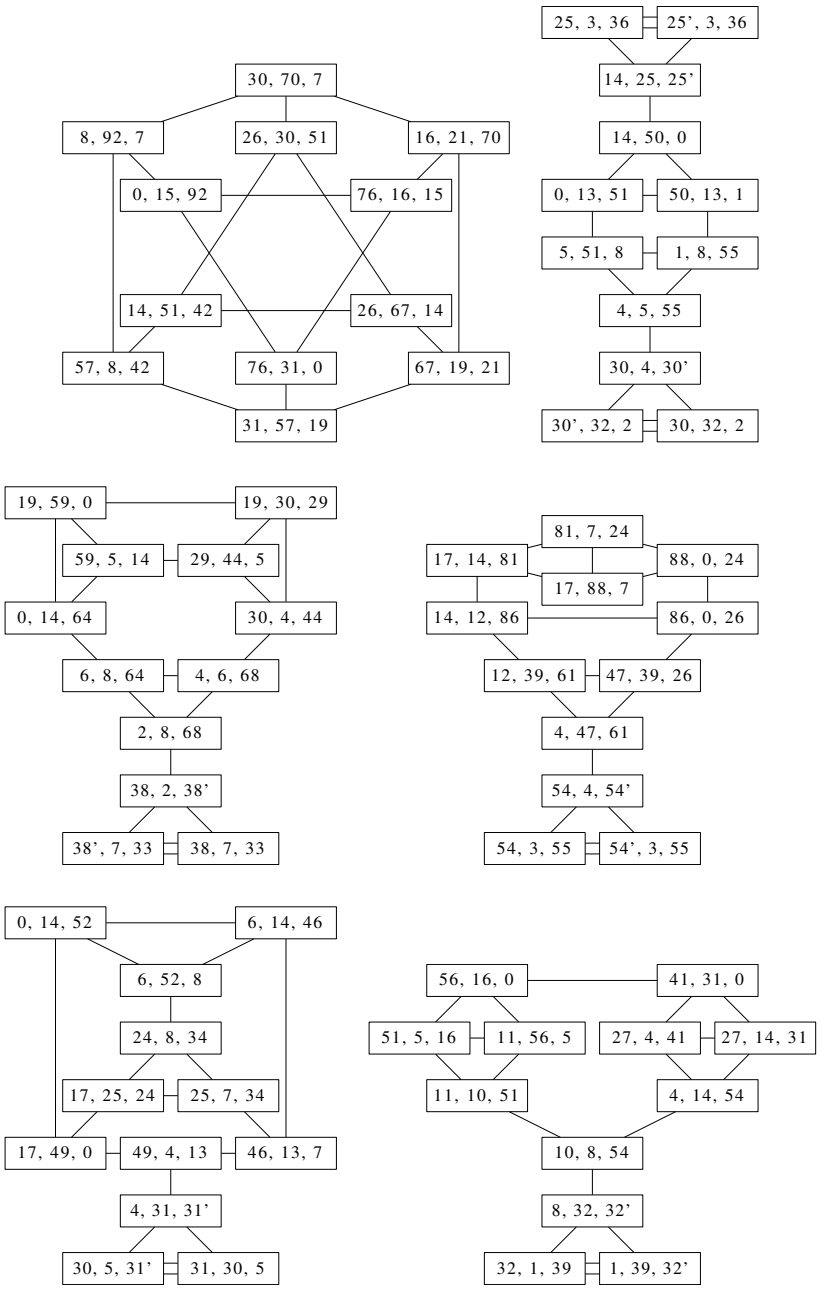


Figure 3.4: The candidate set requires that $h + i + j = \beta$

Since there are 12 points in each of the multigraphs in Figure 3.3, this amounts to 12 linear constraint of the type $h + i + j = \beta$. For our convenience, we allow the weights to be fractions, and let $\beta = 1$. Once we have found fractional weights that satisfy this, we can multiply by the denominators to get integer weights. This gives us a system with 18 variables and 12 equalities. One particular solution to this system would be to choose $w(a) = \frac{1}{3}$ for all elements. It is not difficult to see that this yields a feasible 3PART instance (so it is not nearly-feasible). In order to prevent this from happening, we demand that for all edges h, i, j that do not meet in a single vertex, $h + i + j \neq \beta$. Note that this amounts to $\binom{18}{3} - 12 = 804$ inequalities on the 18 variables. We obtain the following constraints:

$w(a) + w(a') + w(a'') = 1$	For edges a, a', a'' to which a vertex is incident
$w(a) + w(a') + w(a'') \neq 1$	For edges a, a', a'' to which no vertex is incident

To solve this problem, we have used the SMT-solver "Yices", which has the advantage of giving fractional solutions. Details about this are found in appendix A.3. There were solutions for all 14 cubic multigraphs, namely the ones shown in Figure 3.5. We have drawn them as solution-graphs, indicating elements by their weights with a $'$ added to distinguish between different elements with equal weights.


30, 70, 7
8, 92, 7
26, 30, 51
16, 21, 70
0, 15, 92
76, 16, 15
14, 51, 42
26, 67, 14
57, 8, 42
76, 31, 0
67, 19, 21
31, 57, 19
25, 3, 36
25', 3, 36
14, 25, 25'
14, 50, 0
0, 13, 51
50, 13, 1
5, 51, 8
1, 8, 55
4, 5, 55
30, 4, 30'
30', 32, 2
30, 32, 2
19, 59, 0
19, 30, 29
59, 5, 14
29, 44, 5
0, 14, 64
30, 4, 44
6, 8, 64
4, 6, 68
2, 8, 68
38, 2, 38'
38', 7, 33
38, 7, 33
81, 7, 24
17, 14, 81
88, 0, 24
17, 88, 7
14, 12, 86
86, 0, 26
12, 39, 61
47, 39, 26
4, 47, 61
54, 4, 54'
54, 3, 55
54', 3, 55
0, 14, 52
6, 14, 46
6, 52, 8
24, 8, 34
17, 25, 24
25, 7, 34
17, 49, 0
49, 4, 13
46, 13, 7
4, 31, 31'
30, 5, 31'
31, 30, 5
56, 16, 0
41, 31, 0
51, 5, 16
11, 56, 5
27, 4, 41
27, 14, 31
11, 10, 51
4, 14, 54
10, 8, 54
8, 32, 32'
32, 1, 39
1, 39, 32'

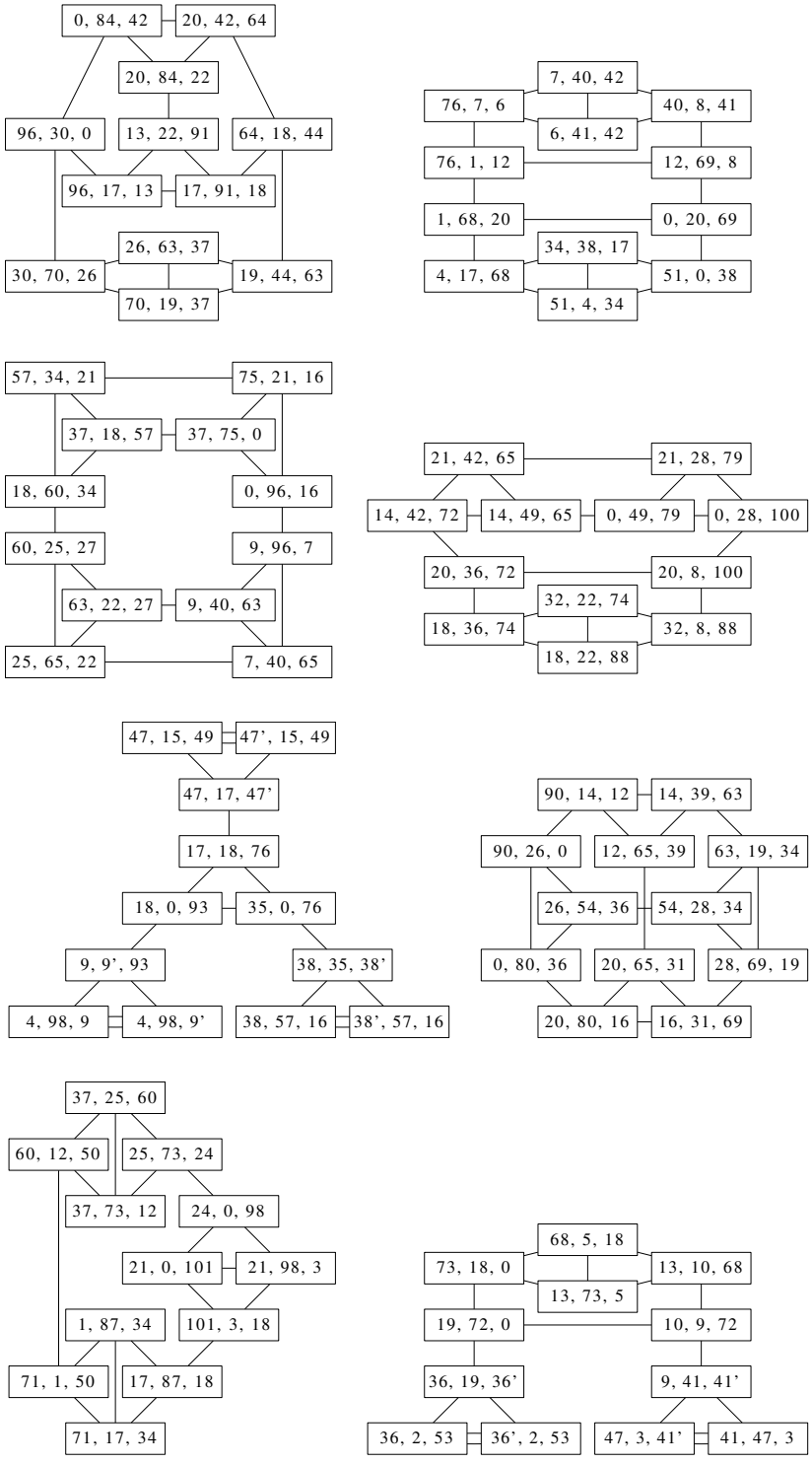


Figure 3.5: 14 nearly feasible instances

3.2.1 Instances that are not locally minimal

The requirement that $w(a) + w(a') + w(a'') \neq 1$ for edges a, a', a'' to which no vertex is incident might seem very strong. One might wonder whether solution-graphs exist for which no such instance can be found. This chapter will show that such graphs exist.

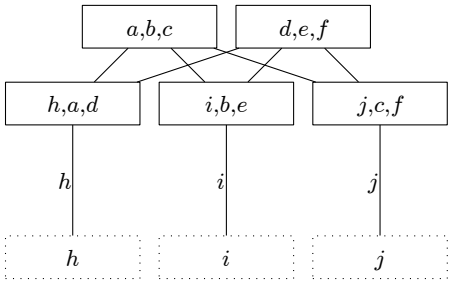


Figure 3.6: The structure of this graph requires that $h + i + j = \beta$

Take a look at Figure 3.6. It contains a solution graph with five candidate sets: $\{a, b, c\}, \{d, e, f\}, \{a, d, h\}, \{b, e, i\}, \{c, f, h\}$. Note that $w(\{a, b, c\}) = w(\{d, e, f\}) = \beta$ by the first two candidate sets. Summing the other three candidate sets yields that weights of all elements (including h, i and j) should sum to 3β. Hence $w(\{h, i, j\}) = \beta$ and $\{h, i, j\}$ forms a candidate set which is not in the graph.

To find an example that has this graph as a solution graph, we remove this one requirement. One such example is:

$0, 3, 14, 15, 17, 18, 19, 21, 24, 27, 28, 31, 56, 71, 74, 75, 77, 79, 83, 84, 112, 132, 133, 143$

Two corresponding solution graphs are shown in Figure 3.7. Note that only

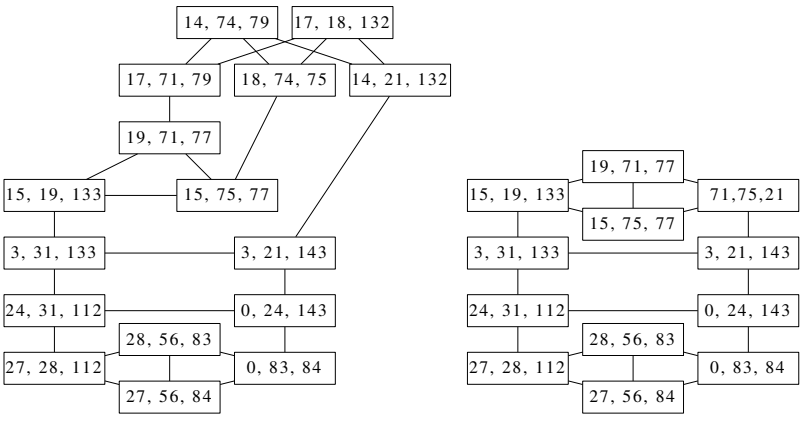


Figure 3.7: The structure of this graph requires that $h + i + j = \beta$

the first contains the structure of Figure 3.6. The latter shows that the instance is not locally minimal, since $|V| = 2q$ does not hold. This property is true in general: any graph containing the structure of Figure 3.6, does not correspond to the solution graph of a locally minimal nearly-feasible instance. To see

this, assume 3.6 is part of the solution graph of some locally minimal nearly-feasible solution. Take the sets $\{a, b, c\}$ and $\{d, e, f\}$ twice, $\{h, i, j\}$ once, and $\{a, d, h\}, \{b, e, i\}, \{c, f, h\}$ zero times. Leave the rest of the solution as it was, and one may check that this yields a solution to which the solution graph has fewer vertices.

3.3 Smallest nearly-feasible instance

From the previous section, we have learned the least q for which there is a nearly-feasible instance, namely $q = 6$. In this section, we will use brute-force search to find nearly-feasible instances having $q = 6$ in order to find the smallest nearly-feasible instance. By "smallest", we mean that its highest weight is minimal among all nearly-feasible instances with $q = 6$. In this section, we give an instance with weights ≤ 10 and $q = 6$. In order to prove that it is the smallest, we have found all instances with $q = 6$ and weights ≤ 10.

Testing whether the instance has a solution to (3.3) for $M = 1$ and $M = 2$ is slow, however. Therefore, we have used the following observations to reduce running-time to eight hours on a 1.6GHz netbook.

1. Every permutation of an instance is also an instance, and adding a constant to all weights of an instance will also yield an instance. Therefore, we only generate instances where the weights form a non-decreasing sequence starting at 0.

2. The sum of all weights must be a multiple of 6, since β is integer.

3. If m is the highest weight in an instance in which the sum per set is β, replacing weight $w(i)$ with $m - w(i)$ for all elements i creates another instance in which the sum per set is $3m - \beta$. Therefore, we only generate instances where $\beta \leq \frac{3}{2}m$.

4. In a nearly-feasible instance of 18 elements, every element occurs in at least 2 candidate sets. Therefore, we only proceed with problems for which this holds.

Using these conditions, 701827 instances remain to be checked. To determine whether (3.3) has a solution, we use simplex to find an answer to (3.3). If (3.3) does not have a solution, or the solution returned had $M = 1$, then the instance was not nearly-feasible. If the solution returned was a $M = 2$ solution and there is a candidate set $\vec{y}_A = M$, the instance is not nearly-feasible. In all other cases, we check whether there are no $k - 1$ disjoint candidate sets to ensure that the instance is not feasible. The remaining instances all turned out to be nearly-feasible instances (so they had a solution in which $M = 2$). These instances were:

- $0, 0, 1, 1, 1, 2, 2, 2, 4, 4, 4, 5, 5, 5, 8, 8, 10, 10$
- $0, 0, 1, 1, 1, 2, 3, 3, 4, 4, 4, 4, 4, 6, 6, 9, 10, 10$
- $0, 0, 1, 1, 2, 2, 2, 2, 4, 4, 4, 5, 5, 5, 8, 8, 9, 10$
- $0, 1, 1, 1, 2, 2, 2, 4, 4, 4, 4, 4, 5, 5, 5, 8, 10, 10$
- $0, 1, 1, 1, 2, 2, 3, 3, 3, 4, 4, 4, 6, 6, 6, 6, 10, 10$

- $0, 1, 1, 1, 2, 3, 3, 3, 4, 4, 4, 4, 4, 6, 6, 6, 10, 10$
- $0, 1, 1, 2, 2, 2, 2, 4, 4, 4, 4, 4, 5, 5, 5, 8, 9, 10$

There are more instances than these seven. We mentioned previously that every permutation of an instance is also an instance. Apart from this, we can also apply the trick mentioned as item 3. This yields the following seven instances:

- $0, 0, 2, 2, 5, 5, 5, 6, 6, 6, 8, 8, 8, 9, 9, 9, 10, 10$
- $0, 0, 1, 4, 4, 6, 6, 6, 6, 6, 7, 7, 8, 9, 9, 9, 10, 10$
- $0, 1, 2, 2, 5, 5, 5, 6, 6, 6, 8, 8, 8, 8, 9, 9, 10, 10$
- $0, 0, 2, 5, 5, 5, 6, 6, 6, 6, 6, 8, 8, 8, 9, 9, 9, 10$
- $0, 0, 4, 4, 4, 4, 6, 6, 6, 7, 7, 7, 8, 8, 9, 9, 9, 10$
- $0, 0, 4, 4, 4, 6, 6, 6, 6, 6, 7, 7, 7, 8, 9, 9, 9, 10$
- $0, 1, 2, 5, 5, 5, 6, 6, 6, 6, 6, 8, 8, 8, 8, 9, 9, 10$

In particular, we proved:

Theorem 7. *Every nearly-feasible instance with $q \leq 6$, has an element with weight* 10 *or higher.*

By our examples, this theorem is tight.

Chapter 4

Variations on nearly-feasible instances

In this chapter, we give variations on nearly-feasible instances. Section 4.1 looks at variations on the way nearly-feasible is defined with respect to (3.3). Section 4.2 extends the notion "nearly-feasible" to different NP-complete problems. Section 4.3 looks at bin-packing games.

4.1 Different relaxations for 3PART

So far, we have focused on nearly-feasible instances. That is, instances where there exist solutions to (3.3) for $M = 2$, but not for $M = 1$. Note that if (3.3) has a solution with some M, then multiplying the corresponding solution $\vec{y}$ with some constant n yields a solution $(\vec{y} \cdot n, M \cdot n)$. As variation on the theme of nearly-feasible instance, we will look at instances of 3PART where a solution with $M = 3$ exists, but not with $M = 1$. We stumbled on one such instance while looking for nearly-feasible instances:

$$0, 0', 0'', 1, 1', 2, 3, 3', 4, 4', 4'', 4''', 4'''', 6, 6', 9, 10, 11$$

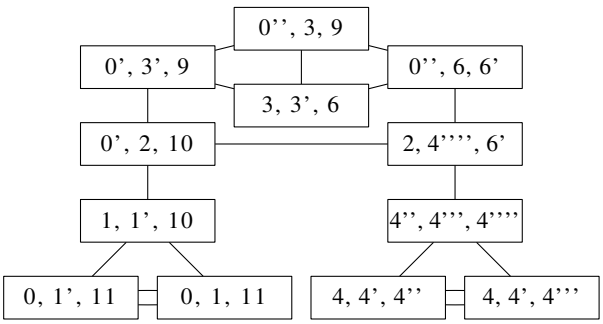


Figure 4.1: A nearly feasible instance that has solutions with $M = 3$

This instance has a solution with $M = 2$ and with $M = 3$. Taking linear combinations of these solutions yields a solution for every $M > 1$. Figure 4.1

shows that the instance is nearly-feasible. A solution with $M = 3$ is shown in table 4.1. Note that the corresponding solution has two components: we have suggestively placed a line in table 4.1 separating the elements with weight 4 from the others. However, neither of the components yields an instance to 3PART (there is no integer q to get $|A| = 3q$).

$C = \{a,$	$a',$	$a''\}$	y_C
0	11	1	2
0	11	1'	1
$0'$	2	10	2
$0'$	6	$6'$	1
$0''$	3	9	1
$0''$	$3'$	$6'$	1
$0''$	6	$6'$	1
1	$1'$	10	1
$1'$	2	9	1
3	$3'$	6	1
3	$3'$	$6'$	1
4	$4'$	$4''$	1
4	$4'$	$4'''$	1
4	$4'$	$4''''$	1
$4''$	$4'''$	$4''''$	2

Table 4.1: A solution for (3.3) where $M = 3$

Another variation would be to look at instances with a $M = 3$ solution, but no $M = 2$ solution (implying that there is no $M = 1$ solution either). The previous instance allows us to construct such an instance: we know that the 13 numbers $0, 0', 0'', 1, 1', 2, 3, 3', 6, 6', 9, 10, 11$ can yield an $M = 3$ solution. It is also evident that there is no $M = 2$ solution, since $13 \neq q \cdot 3$. To create our instance, we repeat and scale these 13 numbers three times, such that no candidate sets are formed between partitions. We make use of the following theorem:

Theorem 8. *Let A and A' be (disjoint) sets and let w and w' be corresponding weight functions. Let $\mathcal{C}$ and $\mathcal{C}'$ be the corresponding candidate sets according to (3.1), with β and β' as sum per set respectively. In addition, we require $w'(a') \neq \beta'/3$ for all $a' \in A'$ and $w(a) \neq \beta/3$ for all $a \in A$. Let $A'' = A \cup A'$ and w'' be defined:*

$$w''(i) = \begin{cases} 3w(i) + 3\beta \cdot \beta' & i \in A \\ 9\beta \cdot w'(i) + \beta & i \in A' \end{cases}$$

Then the candidate sets $\mathcal{C}''$ of A'', w'' are exactly $\mathcal{C}'' = \mathcal{C} \cup \mathcal{C}'$ with $\beta'' = 9\beta' \cdot \beta + 3\beta$ as sum per set.

Proof. $\mathcal{C} \cup \mathcal{C}' \subseteq \mathcal{C}''$ is immediate from the observation that the sum per set in A'', w'' is $\beta'' = 9\beta' \cdot \beta + 3\beta$.

To show $\mathcal{C}'' \subseteq \mathcal{C} \cup \mathcal{C}'$, assume not: $C \in \mathcal{C}''$ contains elements from both A and A'. We distinguish two cases: C either contains one or two elements from A.

In the first case, C contains one element $a \in A$, and two $a'_1, a'_2 \in A'$. The two elements have original weights $w'(\{a'_1, a'_2\}) = n \in \mathbb{N}$, so $w''(\{a'_1, a'_2\}) = 9n \cdot \beta + 2\beta$. Using that all sets should sum to β'', and the definition of w'':

$$w''(a) = \beta'' - w''(\{a'_1, a'_2\}) = 9\beta' \cdot \beta - 9n \cdot \beta + \beta$$
$$w''(a) = 3w(a) + 3\beta \cdot \beta'$$

Solving for $w(a)$:

$$w(a) = \frac{6\beta' - 9n + 1}{3} \cdot \beta \tag{4.1}$$

Using $0 \leq w(a) \leq \beta$ we get $0 \leq 6\beta' - 9n + 1 \leq 3$. This means $2\beta' - 2/3 \leq 3n \leq 2\beta' + 1/3$. Since n is integer, $n = 2\beta'/3$. Filling in n into 4.1 yields $w(a) = \frac{1}{3} \cdot \beta$, a contradiction.

In the second case, C contains one element $a' \in A'$ and two $a_1, a_2 \in A$. Using that $0 \leq w(a_1) \leq \beta$ and $0 \leq w(a_2) \leq \beta$ we get:

$$6\beta \cdot \beta' \leq w''(\{a_1, a_2\}) \leq 6\beta \cdot \beta' + 6\beta$$

Again using that all sets should sum to β'':

$$3\beta' \cdot \beta \leq w''(a') \leq 3\beta' \cdot \beta + 3\beta$$

By definition $w''(a') = 9\beta \cdot w'(a') + \beta$ for some integer $w'(a')$. Solving for $w'(a')$ yields $w'(a') = \beta'/3$, a contradiction.

Since both cases yield a contradiction, we conclude $\mathcal{C}'' = \mathcal{C} \cup \mathcal{C}'$. □

Trying to find smaller examples, we have looked for sets with $|A|$ not a factor of 3, while requiring that the candidate sets $\mathcal{C}$ yield a solution to 3.3. This implies that the corresponding M is a multiple of 3. In order to be able to apply Theorem 8, we require $w(a) \neq \beta/3$. Two of the smallest sets we found were:

- A set of 10 elements: $0, 1, 1, 2, 2, 4, 4, 4, 5, 7$
- A set of 11 elements: $0, 0, 1, 1, 2, 2, 4, 4, 5, 7, 7$

Together, applying Theorem 8, this yields an instance with 10+11 = 21 elements with no $M = 1$ or $M = 2$ solution, but with an $M = 3$ solution.

In an attempt to find even smaller instances, we in addition required that $w(a) + w(a') \neq 2\beta/3$ for all $a, a' \in A$. We found:

- 13 elements: $0, 0, 1, 1, 2, 2, 4, 4, 7, 7, 11, 13, 13$
- 14 elements: $0, 0, 1, 1, 2, 2, 4, 4, 7, 7, 7, 11, 11, 13$

By our requirement, we can add elements with weight $\beta/3$ to these instances without making them feasible. This yields an instance of 18 elements which has an $M = 3$ solution, but no $M = 2$ solution:

$$5, 5, 5, 5, 5, \quad 0, 0, 1, 1, 2, 2, 4, 4, 7, 7, 11, 13, 13$$

Note that we do not know whether the instance with these properties is smallest in terms of β or weights.

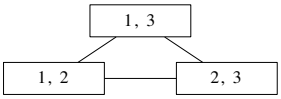


Figure 4.2: A nearly-feasible solution for set cover

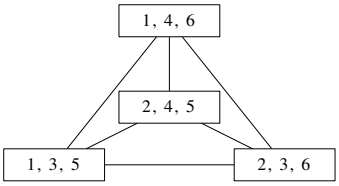


Figure 4.3: A nearly-feasible solution for Exact cover by 3-sets

4.2 Nearly-feasible instances for different decision problems

In Section 2.2.1 we have seen various decision problems. In this section, we take another look at some of them and give a nearly-feasible instance with the least number of elements. For all these problems, $\mathcal{C}$ is defined in some way - a variation of (2.1) or (3.1) - and the instance is called nearly-feasible if:

$$\sum_{C \ni e} \vec{y}_C = M \quad \text{for all } e \in A,$$
$$\vec{y} \in \{0, \ldots, M\}^{\mathcal{C}}.$$

has a solution for $M = 2$, while for $M = 1$ there is no such $\vec{y}$. Note that this general definition is identical to our definition in the case of 3PART. We will also give solution-graphs according to our previous definition.

Set-cover

A set cover problem over elements A is directly defined by candidate sets: $\mathcal{C} \subseteq \mathcal{P}(A)$. The smallest instance contains 3 elements, $1, 2, 3$, with:

$$\mathcal{C} = \{\{1, 2\}, \{2, 3\}, \{1, 3\}\}$$

Exact cover by 3-sets

The problem is the same as in the previous case, but this time we demand that for every $C \in \mathcal{C}$ we have $|C| = 3$. With the 6 elements: $1, 2, 3, 4, 5, 6$ and $\mathcal{C} = \{\{1, 3, 5\}, \{2, 3, 6\}, \{2, 4, 5\}, \{1, 4, 6\}\}$ we obtain a nearly-feasible instance.

3-dimensional matching

This time we have three sets: X, Y, Z. We require X, Y, Z to be disjoint and the items to be coverd are $A = X \cup Y \cup Z$. This case, $\mathcal{C}$ is restricted to sets with

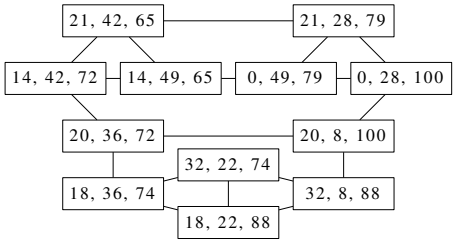


Figure 4.4: A nearly-feasible solution for the Numerical 3-dimensional matching

exactly one element from X, Y and Z:

$$\mathcal{C} \subseteq \{\{s_1, s_2, s_3\} \mid (s_1, s_2, s_3) \in S_1 \times S_2 \times S_3\}$$

Our previous example provides us with a 3-dimensional matching: $X = \{1, 2\}$, $Y = \{3, 4\}$, $Z = \{5, 6\}$ and $\mathcal{C} = \{(1, 3, 5), (2, 3, 6), (2, 4, 5), (1, 4, 6)\}$.

Numerical 3-dimensional matching

Recall the definition of Numerical 3-dimensional matching:

Definition (N3DM)**.** *The Numerical 3-dimensional matching (*N3DM*) consists of* 3 *sets* S_1, S_2, S_3 *containing* q *distinct elements each, a weight function for each set:* $w_i : S_i \to \mathbb{N}$*, and a bound* β*. Candidate sets* $\mathcal{C}$ *are given by:*

$$\mathcal{C} = \{\{s_1, s_2, s_3\} \mid (s_1, s_2, s_3) \in S_1 \times S_2 \times S_3,\ w_1(s_1) + w_2(s_2) + w_3(s_3) = \beta\}$$

The objective is to find a set $M \subseteq \mathcal{C}$ *such that* M *is a partition of* $S_1 \cup S_2 \cup S_3$*.*

All observations on 3PART hold for the Numerical 3-dimensional matching problem, so we cannot hope to find an instance in which $q < 6$. Indeed, some of the graphs in Figure 3.5 (namely all 3-colorable ones) provide us with nearly-feasible instances for the Numerical 3-dimensional matching. For example, these weights yield a nearly-feasible solution (Figure 4.4):

$$w_1 : 0, 14, 18, 20, 21, 32$$
$$w_2 : 8, 22, 28, 36, 42, 49$$
$$w_3 : 65, 72, 74, 79, 88, 100$$

4.3 Bin packing games

We wish to apply some of the results we found on nearly-feasible instances to bin packing games. We addressed the question whether the ϵ-core is non-empty in Section 2.3. Recall that we can answer this question by calculating the optimal value for (2.4) and (2.5). For bin packing games in which all weights are $> \beta/3$, there is a game with an empty ϵ-core if and only if $\epsilon < \frac{1}{7}$. Looking at games with weights $> \beta/3$ implies that at most two items can be packed per bin. In

relation to 3PART, we look at weights in $(\beta/4, \beta/2)$, which implies that at most three items can be packed per bin.

Take a nearly-feasible 3PART problem given by $3q$ elements A and weights $w : A \to \mathbb{N}$, with bound β. There is a corresponding bin packing problem with elements A, weights $w'(a) = w(a) + M$, q bins and bound $\beta + 3M$. If M is sufficiently large, then at most three elements fit in each bin. There is a vector $\vec{x}^*$ (which is an optimal solution to (2.4)) such that every element gets packed:

$$\sum_{S \in \mathcal{C}} w'(S) \cdot \vec{x}^*_S = w'(A)$$

This follows from taking the nearly-feasible solution. We also know that, since the 3PART instance was not feasible, there is some element that does not get packed in an optimal solution to (2.5). Suppose a' is one such element, then $v(N) \leq w'(A) - w'(a)$.

The ϵ-core is nonempty if and only if $\epsilon \geq (w'(A) - v(N))/w'(A)$. Suppose the ϵ-core is nonempty, then:

$$\begin{aligned}
\epsilon &\geq \frac{w'(A) - v(N)}{w'(A)} \\
&\geq \frac{w'(A) - w'(A) + w'(a)}{w'(A)} = \frac{w'(a)}{(\beta + 3M) \cdot q} \\
&\geq \frac{M}{(\beta + 3M) \cdot q}
\end{aligned}$$

If we take the limit of $M \to \infty$, we see that the ϵ-core is nonempty for $\epsilon < \frac{1}{3q}$. Taking a smallest nearly-feasible instance yields that for $\epsilon \leq 1/18$ there is a game with an empty core. This proves that for $\epsilon < 1/18$, there are games with an empty ϵ-core with all weights in $(\beta/4, \beta/2)$.

We can do better than $1/18$ by looking at elements and weights that do not form a 3PART instance. Let B be a set of 3 bins of size $\beta = 9 + 3M$, and A a set of 10 elements of respective size:

$$M, 1 + M, 1 + M, 2 + M, 2 + M, 4 + M, 4 + M, 4 + M, 5 + M, 7 + M$$

These are 10 elements taken from Section 4.1. We will use Theorem 8 to prove that it is not possible to fill 3 bins with 9 out of 10 elements.

Proof. Observe that no element of A has weight $\beta/3$. We create a new 3PART instance A', w' by taking disjunctive sets A_1, A_2, A_3. Note that $|A_1| + |A_2| + |A_3| = 30$, hence we have a 3PART instance. The weights are chosen such that, by Theorem 8, for each candidate set C: $\exists i.\ C \subset A_i$. From this, it follows that there is no solution to the 3PART instance. By Theorem 5, we cannot even pick 9 disjoint candidate sets. This means that for some i, we cannot pick 3 disjoint candidate sets of A_i. Hence, we cannot pick 3 disjoint candidate sets of A. □

We could, of course, have proven that it is impossible to fill 3 bins with 9 out of 10 elements by hand. This proof, however, is more general: it does not matter what the weights of our 10 elements were, as long as no element has weight $\beta/3$.

Chapter 5

Conclusion and discussion

When a decision problem is NP-hard, we have little hope of finding an efficient (polynomial-time) algorithm for it. Looking at a related decision problem instead, such as the LP relaxation of an ILP, might solve the original problem in some cases. A question that arises is: when does the relaxation fail to solve the original problem.

In this research, we attempted to find examples of problem instances that are infeasible, while the relaxation has a solution. In order to do so, we have looked at the abstract structure of such instances, and used different types of computer support to find the corresponding instances.

In this research, we have looked at different relaxations of 3PART. There has been a focus on nearly-feasible instances, which are shown to exist. In particular, the underlying structure was that of a cubic multigraph. In order to find nearly-feasible instances, we have used graph-generation tools and SMT-solvers.

One such nearly-feasible instance is given by the following weights:

$$0, 0, 1, 1, 1, 2, 2, 2, 4, 4, 4, 5, 5, 5, 8, 8, 10, 10$$

We know that this instance is the smallest in terms of number of items. Out of all instances with 18 items, the largest weight in it is smallest in this instance. This is the main result of [9].

Apart from looking for nearly-feasible instances, we have found examples for different variations. We have extended the notion of "nearly-feasible" to the set-cover, set-cover by 3-sets, 3-dimensional matching, and numerical 3-dimensional matching problem. For those problems, we were able to find the smallest nearly-feasible instance in terms of number of items.

We have also looked at different relaxations of 3PART. For those, we do not know whether we have found a smallest instance. However, one of these relaxations provided us with a result to bin packing games.

One of the reasons we started our investigation of 3PART, is that it arose as a question in bin packing games. We have seen that for $\epsilon < 1/9$, there is a bin packing game with an empty ϵ-core with all weights in $(\beta/4, \beta/2)$. The corresponding bin packing game has 10 elements. This game corresponded to a variation of 3PART.

Appendix A

Implementation details

This chapter is about everything that has to do with computers. It is included to ensure that all results in this thesis are reproducible. This chapter assumes the reader to have some experience in Haskell. Not all code is included in this report. Should you be interested in running the code on your machine, please contact me at `sjcjoosten@gmail.com` and I will send the required sources.

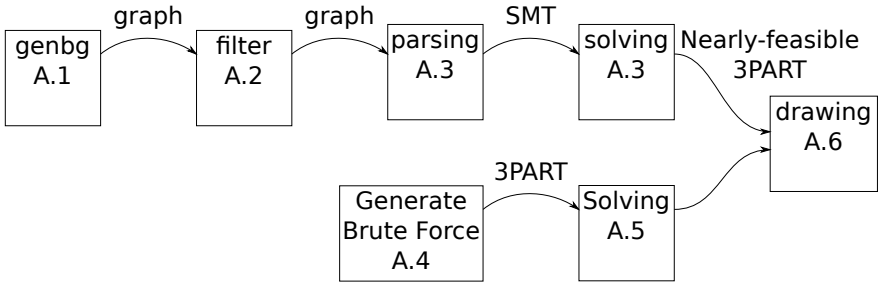


Figure A.1: Combining computer programs, and the sections describing them.

An overview of all code is shown in Figure A.1. Connected cubic multigraphs are generated with the "`genbg`" package from the "`nauty`" program. An alternative way to get connected cubic multigraphs, is by using a `c++` program written by Twan van Laarhoven. Both ways are described in Section A.1. The resulting graphs can be filtered: we throw away graphs having $q-1$ disjunct points. After this step we are left with the graphs shown in Figure 3.3. The next step is finding the actual instance. This involves calling an SMT-solver. The call to the SMT-solver is described in Section A.3.

We can also generate 3PART instances without using a graph-structure. This is described in Section A.4. Being given an instance, we are be able to test whether it is nearly-feasible. The test whether it is feasible (it should not be), is described in Section A.5. Finding a solution to (3.3), and drawing it as a graph, is described in Section A.6.

A.1 Obtaining connected cubic multigraphs

Instructions for installing the "`nauty`" program can be found on the nauty page: `http://cs.anu.edu.au/ bdm/nauty/`. The program we use is called "`genbg`", which is a program for generating bipartite graphs. We generate a graph with $3q$ vertices in one partition, which will represent the edges of our multigraph. The $2q$ vertices in the other partition will represent the vertices. Since we are looking for cubic multigraphs, we require that all vertices in the second partition have degree 3. Since an edge is given by 2 vertices, we require that all vertices in the first partition have degree 2. Finally, we require that the entire graph is connected. The call to "`genbg`" now looks like this:

```
genbg -c -D2:3 -d2:3 18 12 -a
```

The switch `-c` requires connectedness, `-D2:3 -d2:3` is to set the degrees, `18 12` is to set the number of vertices. In order to get readable output, we use the switch `-a`. The output is a list of graphs, separated by newlines, of which the first one is represented as:

```
123,124,567,568,9AB,CDE,FGH,9AI,3CD,7BF,4GH,8EI.
```

We interpret this as a multigraph as follows. All $2q$ vertices are listed, separated by a comma. A vertex is represented by the edges to which it is connected, which are labelled by single characters.

An alternative way to get connected cubic multigraphs, is by using a `c++` program written by Twan van Laarhoven. I have modified the code slightly in order to have it produce the same output format as "`genbg`". The code of Twan has the advantage of being slightly faster, and produces larger graphs as output ("`genbg`" has built-in limits). It is 300 lines (7 pages) of code, and is available on request.

A.2 Filtering graphs

To filter graphs, the graph is read, parsed, and we check whether there are $q-1$ independent vertices. To check whether there are $q-1$ independent vertices, we generate all sets of $q-1$ independent vertices. As in the previous section, a vertex is represented by a set of edges to which it is connected. All sets of `i` independent vertices are given by `allSets i`. We use the auxiliary function `unforbidden` to determine which vertices are independent of a given set. The code is as follows:

```
import Data.List.Ordered (nubSort)

hasDist::[[Char]]->[[[Char]]]
hasDist inp = allSets (q - 1)
 where
  q = div (length $ (nubSort.concat) inp) 3
  allSets 0     = [[]]
  allSets i = [u:r | r<-allSets (i-1), u<-unforbidden r]
  unforbidden r = [a|a<-inp
                    , null r || a < head r
```

```
                    , and $ map (noIsect a) r]
noIsect as bs = null [a|a<-as,a `elem` bs]
```

A.3 Obtaining 3PART instances from graphs

As mentioned earlier, we create a SMT instance where three variables sum to 1 if and only if they represent edges that share a vertex. To determine whether edges h, i, j share a vertex, we use `together h i j`. To build an SMT-term, we use operators to combine terms. For instance, the operator `OpAnd` takes a list of terms, which should all hold in order for the term to be true. The operators `OpEq` and `OpNeq` take two numerical terms and requires both to be equal or unequal. The operator `OpAdd` takes a list of numerical terms and returns its sum. Constants are added via the function `c`.

The following SMT-term (over the variables `w i` for `i <- elements`) is converted as input to our SMT-solver:

```
OpAnd ([(if together h i j then OpEq else OpNeq)
            (c 1)
            (OpAdd [w h,w i,w j])
        | h <- elements, i <- elements
        , i > h, j <- elements, j > i
        ] ++ -- trivialities:
        [between (c 0) (c 1) (w i)|i <- elements])
where
 between minbound maxbound tm
  = OpAnd [OpLeq tm maxbound, OpGeq tm minbound]
```

The function `together`, and the set `elements` both depend on the input-graph. There is an ordering on `elements`, such that we can remove duplicates (`i>h`, `j>i`) in a convenient way.

A.4 Obtaining 3PART instances

In the previous section we generated instances by using a solution graph. In this section we generate instances by simply trying out different weights. Since we are looking for nearly feasible instances, we apply some of the criteria mentioned in Section 3.3. These criteria are applied one by one, rewriting the program along the way.

To generate all nearly-feasible instances with $n = 3q$ elements having weights in $[0, m]$.

```
instances :: Int -> Int -> [[Int]]
instances 0 _ = [[]]
instances n m
 = [w:ws | w<-[0..m], ws<-instances (n-1) m]
```

However, we actually want instances in ascending order, so we may change the last line:

```
instances n m
 = [w:map ((+) w) ws
```

```
    | w<-[0..m], ws<-instances (n-1) (m-w) ]
```

We want to say something about the total sum s. It should, for one thing, be divisible by q. Therefore, we want to be able to generate all instances on n elements, weights in $[0, m]$ and total sum s.

```
instWithSum :: Int -> Int -> Int -> [[Int]]
instWithSum 0 _ 0 = [[]]
instWithSum n m s
  = [w:map ((+) w) ws
     | w<-[max 0 (s - m*(n-1))..min m (div s n)]
     , ws<-instWithSum (n-1) (m-w) (s-n*w)]
```

Using this new function, we can rewrite our old "`instances`" to get only those instances where $\beta \leq \frac{3}{2}m$, such that β is integer.

```
instances :: Int -> Int -> [[Int]]
instances n m
  = concat [instWithSum n m s
            | beta<-[m..div (3*m) 2]
            , let s=div (beta*n) 3]
```

In generating the instances, we also required the maximum value m to be attained, and the minimum 0 to be used.

A.5 Obtaining a solution to 3PART

There are two ways to find a solution to 3PART. One way is by applying "`hasDist`" mentioned earlier to the candidate sets (let "`Char`" be the type of an element). Another way is by implementing (2.3) we saw in Section 2.2.2. We use an SMT-solver as in Section A.3. For a list of `weights`, the SAT-clause is as follows:

```
OpAnd ( [OpOr [p i | (s,i)<-csets, elem v s]
         | v<-elements] ++
         concat
         [[OpNot (OpAnd [(p j),(p i)])
                        | (s2,i)<-csets
                        ,i<j ,overlap s1 s2]
         | (s1,j)<-csets
         ] )
where
 csets :: [([Int],Int)]
 csets = zip candSets [1..]
 candSets
  = [[a,b,c]
     | (w1, a)<-weights'
     , (w2, b)<-drop a weights'
     , (w3, c)<-drop b weights'
     , w1+w2+w3==div (sum weights * 3)
                     (length weights)]
 weights' = zip weights [1..]
 elements = map snd weights'
```

A.6 Drawing a nearly-feasible solution

To obtain the nearly-feasible solution, we used a package from "`Hackage`" called "`Numeric.LinearProgramming`". The operator `:==:` ensures sum of variables on the left hand side sum to the numeric value indicated by the right hand side. The operator `:&:` indicates that the sum of variables on the left hand side should be between the bounds indicated by the value pair $(\min, \max)$ on the right hand side. On the left hand side, all variables (in our case "`sets`") are multiplied by a constant indicated by the list values, and summed. As an example, suppose we have three variables x_1, x_2, x_3, then $x_1 + 2 \cdot x_3$ would be expressed by the list $[1, 0, 2]$.

The constraints are as follows:

```
[ [if (elem v c) then 1 else 0 | c<-sets] :==: 1
| v <- [0..(k-1)] ]
++ [ [if (c==c') then 1 else 0 | c<-sets] :&: (0,1)
   |c' <- sets]
```

In most cases, we obtained a nearly-feasible solution whenever there was one.

Drawing the solution as a solution graph was done using the candidate sets for which the corresponding variable was > 0. To draw a graph, we converted a nearly-feasible solution to a "`neato`" input file, and use "`neato`" to draw the solution as a graph. In order to get the alignment as in this thesis, we change this file by hand and enter the positions manually.

We give one such "`neato`" file as an example. The set-cover problem instance of Figure 4.2 was generated using the following file:

```
digraph g {
  node [shape=box pin="true" fixedsize="true"
        fontsize=22 height=0.55 width=1.75];
  edge [len=4 dir=none];
  n0[label="1, 2" pos="0,0"];
  n1[label="2, 3" pos="3,0"];
  n2[label="1, 3" pos="1.5,1"];
  n0->n1;n0->n2;n1->n2;
}
```

Bibliography

[1] Stephen A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the third annual ACM symposium on Theory of computing*, STOC '71, pages 151–158, New York, NY, USA, 1971. ACM.

[2] M. E. Dyer and A. M. Frieze. The solution of some random np-hard problems in polynomial expected time. *Journal of Algorithms*, 10(4):484 – 489, 1989.

[3] Ulrich Faigle and Walter Kern. On some approximately balanced combinatorial cooperative games. *Mathematical Methods of Operations Research*, 38:141–152, 1993. 10.1007/BF01414210.

[4] Ulrich Faigle and Walter Kern. Approximate core allocation for binpacking games. *SIAM J. Discret. Math.*, 11:387–399, August 1998.

[5] Sndor P. Fekete, Jrg Schepers, Sandor P. Fekete, Tu Berlin, Or P. Fekete, and J Org Schepers. New classes of fast lower bounds for bin packing problems. In *Mathematical Programming*, pages 11–31, 2001.

[6] M. R. Garey. Complexity results for multiprocessor scheduling under resource constraints. *SIAM Journal of Computing*, 4:397–411, 1975.

[7] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1979.

[8] Bissan Ghaddar, Miguel Anjos, and Frauke Liers. A branch-and-cut algorithm based on semidefinite programming for the minimum k-partition problem. *Annals of Operations Research*, pages 1–20, 2008. 10.1007/s10479-008-0481-4.

[9] Sebastiaan JC Joosten and Hans Zantema. LP relaxation of 3-parition instances. *submitted*.

[10] R. M. Karp. Reducibility among Combinatorial Problems. *Complexity of Computer Computations*, 1972.

[11] W. Kern and X. Qiu. Improved taxation rate for bin packing games. In A. Marchetti-Spaccamela and M. Segal, editors, *First International ICST Conference on Theory and Practice of Algorithms in (Computer) Systems, TAPAS 2011, Rome, Italy*, volume 6595 of *Lecture Notes in Computer Science*, pages 175–180, Berlin, 2011. Springer Verlag.

[12] Donald E. Knuth. Dancing links. Technical report, Stanford University, November 2000.

[13] Jeroen Kuipers. Bin packing games. *Mathematical Methods of Operations Research*, 47:499–510, 1998. 10.1007/BF01198407.

[14] L. A. Levin. Universal sequential search problems. *Probl. Peredachi Inf.*, 9:115–116, 1973.

[15] Silvano Martello and Paolo Toth. *Knapsack problems: algorithms and computer implementations*. John Wiley & Sons, Inc., New York, NY, USA, 1990.

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

[17] Brendan D. McKay. `nauty` *User's Guide (Version 2.4)*. Australian National University, ACT 0200, Australia, November 2009.

[18] N. J. A. Sloane. Sloane's series A000421: Number of isomorphism classes of connected 3-regular loopless multigraphs of order 2n, 2007.

Index