Dissertation

On the Hardness of Computing Local Optima

Dominic Dumrauf

Paderborn, 9. März 2011

Schriftliche Arbeit zur Erlangung des Grades
Doktor der Naturwissenschaften
an der Fakultät für Elektrotechnik, Informatik und Mathematik
der Universität Paderborn

To My Mother

Abstract

In this thesis, we investigate the *complexity of computing locally optimal solutions* of problems arising in the fields of *game theory* and *optimization.* For our investigation, we use the framework of $\mathcal{PLS}$ (short for "Polynomial-time Local Search"), as introduced by Johnson, Papadimtriou, and Yannakakis [56].

Before presenting our results, we first revisit the framework of $\mathcal{PLS}$ and present the necessary notation in Chapter 2. In Chapter 3, we survey the research on the complexity of computing locally optimal solutions. There, we concentrate on well-known and successful local search heuristics for various problems, where our focus is on worst case complexity along with the existence of *sequences of improving steps of exponential length.* We mostly concentrate on surveying research on congestion games, which sparked the interconnection between local search and game theory.

In game theory, congestion games are a widely accepted model to investigate the behavior and performance of large-scale distributed networks with autonomous participants. The class of *restricted network congestion games* is a subclass of congestion games where for each player there exists a set of edges which he is not allowed to use. Rosenthal's potential function guarantees the existence of a Nash equilibrium, as local minima of the potential function coincide with Nash equilibria; moreover, Rosenthal's potential function is polynomial-time computable. This allows to formulate the problem of computing a Nash equilibrium in a given restricted network congestion game as a local search problem. In Chapter 5, we show that computing a Nash equilibrium in a restricted network congestion game with *two players* is *$\mathcal{PLS}$-complete*, using a tight reduction from MaxCut. The result holds for directed networks and for undirected networks.

From the field of optimization, we investigate the complexity of computing locally optimal solutions of the Maximum Constraint Assignment (in short MCA) problem and of weighted standard set problems. In a nutshell, the MCA problem which we study in Chapter 6 is a local search version of weighted Generalized Maximum Satisfiability on constraints (functions mapping assignments to positive integers) over variables with higher valence. The parameters in (p, q, r)-MCA$_{k\text{-par}}$ simultaneously limit the maximum length p of each constraint, the maximum appearance q of each variable and its valence r; additionally, the set of constraints is k-partite. We focus on hardness results and *show $\mathcal{PLS}$-completeness of* $(3, 2, 3)$-MCA$_{3\text{-par}}$ and $(2, 3, 6)$-MCA$_{2\text{-par}}$, using tight reductions from Circuit/Flip. Our results are *optimal* in the sense that $(2, 2, r)$-MCA is solvable in polynomial time for every $r \in \mathbb{N}$. We also pay special attention to the case of *binary variables* and show that $(6, 2, 2)$-MCA is tight $\mathcal{PLS}$-complete. For our results, we extend and refine a technique from Krentel [67].

Finally, in Chapter 7 we study the complexity of computing locally optimal solutions of *weighted standard set problems* such as SetPacking, SetCover, and many more, as pooled in problems [SP1]–[SP10] in the book of Garey and Johnson [40, page 221ff.]. We show that for most of these problems, computing a locally optimal solution is *already $\mathcal{PLS}$-complete* for a *simple natural neighborhood of size one*. For the local search versions of weighted SetPacking and SetCover, we derive tight bounds for a simple neighborhood of size two. To the best of our knowledge, these are one of the very few $\mathcal{PLS}$ results on local search for weighted standard set problems.

The investigations in this thesis are mainly led by showing hardness results for the local search problems outlined above. Ideally, we would like to *demarcate the tractability* of computing locally optimal solutions for these problems. Moreover, we are interested in *commonalities* between the reductions we present, as well as *potential sources of intractability* in the problems we show hardness for. We discuss these superior questions in Chapter 8 and point out potential directions for further research, by stating various open problems.

Acknowledgements

During the creation of this thesis, I have become deeply indebted to numerous people without whom large parts of this work would not have been possible. First and foremost, I would like to thank my advisor Burkhard Monien for his continuos support and encouragement. His expertise and guidance have been a crucial factor in the entire creation of this work.

The DFG (German Research Foundation) Research Training Group "Paderborn Institute for Scientific Computation" (PaSCo) at the University of Paderborn did provide me with a three year fellowship for my Ph.D. project. During that time, a visit to the research group of Petra Berenbrink at the Simon Fraser University has given me an inspiration that goes beyond the results obtained in Chapter 7. For providing me with this great opportunity, I would like to thank Petra Berenbrink, Burkhard Monien, Robert Preis, and the PaSCo GK. Furthermore, I would like to thank Christian Scheideler for supporting me in the last months of my Ph.D. project.

During the last years, I have been very lucky to learn from and cooperate with excellent researchers. I would especially like to thank my co-authors Tim Süß, Karsten Tiemann, and Tobias Tscheuschner for countless discussions, inspirations, and invaluable feedback. I am thankful to Martin Gairing and Sven Grothklags for help and advice on the first steps of my research. For carefully reading chapters of a preliminary version of this thesis, I would like to thank Martina Hüllmann and Michelle Kloppenburg. I could not have been more lucky than with my excellent office mates Karsten Tiemann and Adrian Ogierman, to whom I could always turn to on scientific and non-scientific topics. I would also like to thank all my colleagues who provided me with help and support. To name the ones that I did not already mention: Ulrich Ahlers, Bernard Bauer, Yvonne Bleischwitz, Robert Elsässer, Rainer Feldmann, Sigrid Gundelach, Henning Meyerhenke, Marion Rohloff, Thomas Sauerwald, Stefan Schamberger, Ulf-Peter Schroeder, and Thomas Tissen.

Mother, your encouragement and faith throughout all these times have always been a huge support for me. It is a wonderful feeling to be gifted with such a great mother! The support and advice of Johannes Berg, Daniel Krause, Christiane Lammersen, and Tim Süß has always been invaluable to me. Even though our roads took different turns, I am very thankful for everything I was allowed to learn from Kristin Sommer. Furthermore, I would like to thank Verena Appelhans, Christian Deutscher, Joachim Gehweiler, André Größer, Anna Kortmann, Andreas Kumlehn, Irmgard Mietusch, Marta Schlossarek, Florian Schoppmann, David Teusner, and Claudia Unterkircher.

Paderborn, March 2011 Dominic Dumrauf

Contents

Chapter 1

Introduction

Imagine the following scenario: It's already noon and it seems like work is progressing at a snail's pace! That pivotal meeting of the board of directors is coming up at two o'clock, including your election to the board of directors. Besides all the bustle at work, the kids have to be picked up from school at five. In the meantime, there remains a lot of unfinished business in your area of responsibility:

1. There is a department moving to a new floor and it is your duty to assign the employees to the available cubicles. There are certain constraints to obey when designing the seating chart. Some employees need to be seated in clusters for collaboration, some require a fax machine nearby, and others just want to be located closest to the coffee machine; apparently, everybody wants his or her own cubical. The priority of a given set of employees moving to the new floor is expressed as a positive integer. This leaves you with the task of finding a pairwise disjoint subset of the set of given sets of employees which has maximum total priority and obeys the above constraints.

2. Since creating seating charts for moving departments is not your sole task, you decide to hand it over to some of your assistants, as you have done before. The assistants all have different levels of expertise and are only productive when working in teams of like-minded colleagues. Assigning a team to a set of tasks incurs a certain cost. All tasks must be completed and for that you are willing to tolerate double execution of a task. This leaves you with the responsibility of finding a least expensive subset of the given teams such that the selected teams execute all tasks.

3. In addition to the above internal affairs, the day-to-day business needs your attention. The set of your clients can be split into three equally large groups, namely consumers, producers, and distributors. They need to be matched in triples such that no triple contains two members of the same group. The contentment of each triple is expressed in a monetary reward which is handed to you upon assignment. Not all combinations are available, as for example a car manufacturer can make little use of a consumer who wants green tea and both parties interact using a broker who is specialized on florists. This leaves you with the task of finding a matching between the three groups which maximizes your personal reward and obeys the given constraints.

4. The above minutiae should not distract you from further fueling your career in the conglomerate that employs you. An additional salary stimulus comes from serving on as many boards of directors as possible. On the downside, each position claims a share of your precious time; for that reason, you decided to take at most k positions. On the upside, each taken position returns some monetary compensation; some positions are linked to one another, as for example a subsidiary company might share some directors with the parent group. This leaves you with the task to single out the at most k positions to aspire in order to maximize your income.

Due to time constraints you cannot afford to tackle each of the above problems individually. Fortunately, you recall an optimization lecture back at the university on the capabilities of WEIGHTED MAXIMUM SATISFIABILITY. The labor-saving approach is now to model each of the above problems as a WEIGHTED MAXIMUM SATISFIABILITY problem, i. e. to express them as a set of weighted clauses over a set of binary variables such that an optimal solution of an instance of WEIGHTED MAXIMUM SATISFIABILITY constructed this way induces an optimal solution of the instance of the corresponding problem you modeled. This approach reduces the workload from constructing four approaches to the four different problems above to creating a single approach for WEIGHTED MAXIMUM SATISFIABILITY. Luckily, decades of research in computer science have provided you with a solver for WEIGHTED MAXIMUM SATISFIABILITY that supplies you with the appropriate answers such that with great chance you will be there in time to pick up the kids from school.

1.1 Optimization Problems, Intractability, and Approximation

The above examples in items 1–4 illustrate that we are surrounded by optimization problems in our everyday life. The relocation of the department in item 1 is an optimization version of a weighted SETPACKING problem (see next subsection or confer [SP3] in the book of Garey and Johnson [40] for a formal definition). Finding a least expensive subset of the given teams such that the selected teams execute all tasks in item 2 is an optimization version of a weighted SETCOVER problem (see next subsection or confer [SP5] in the book of Garey and Johnson [40] for a formal definition). Matching each consumer, producer, and distributor in item 3 such that the reward is maximized is an optimization version of a weighted 3-DIMENSIONALMATCHING problem (confer [SP1] in the book of Garey and Johnson [40] for a formal definition). Raising the salary in item 4 by serving at most k positions on the boards of directors simultaneously, while maximizing the revenue represents an optimization version of a weighted HITTINGSET problem (confer [SP8] in the book of Garey and Johnson [40] for a formal definition). Finally, the WEIGHTED MAXIMUM SATISFIABILITY problem as used above is an optimization version of the well-known SATISFIABILITY problem (see Subsection 1.1.2 or confer [LO1] in the book of Garey and Johnson [40] for a formal definition). Numerous theoretical and practical applications have made both SATISFIABILITY problems subject to intense investigation in the literature.

Identifying and Categorizing Hard Problems

In computer science, the problems outlined in items 1–4 are pooled as *set problems*. Together with the SATISFIABILITY problems described above, they have been under severe investigation for the last decades. The investigation of optimization problems was in large parts led by the quest of finding algorithms which efficiently compute solutions for the respective problems. In computer science, an algorithm is widely accepted as *efficient* when its running time is polynomial in the size of the input; such algorithms are commonly referred to as *polynomial-time algorithms*. A problem is widely regarded as *tractable* if there exists a polynomial-time algorithm which solves the problem. For the set problems and the SATISFIABILITY problems outlined above, there does not exist a polynomial-time algorithm for a single problem up to the point of publication of this thesis; the general belief is that there is indeed little hope for efficient algorithms.

The Class $\mathcal{NP}$ and the Concept of Reductions A *decision problem* is a problem whose solution is either “yes” or “no”. The decision problems which can be derived from the above optimization problems belong to the class of *decision problems* which can be *solved by a polynomial-time bounded non-deterministic Turing machine*; the corresponding class is called $\mathcal{NP}$. On the one hand, $\mathcal{NP}$ contains problems for which polynomial-time algorithms are known such as SHORTESTPATH or LINEARPROGRAM. On the other hand, $\mathcal{NP}$ also includes an extensive list of important problems for which no polynomial-time algorithm is known, such as the TRAVELING SALESMAN PROBLEM, the SATISFIABILITY problem, and decision problems which can be derived from the set problems outlined above. In order to further categorize the problems in $\mathcal{NP}$ by computational complexity, the *concept of* $\mathcal{NP}$*-completeness* was introduced. An $\mathcal{NP}$ problem B is $\mathcal{NP}$*-complete* if for each $\mathcal{NP}$ problem A, every instance of A can be transformed into an instance of B in polynomial time such that for each instance of A, “yes” is a solution if and only if “yes” is a solution of the transformed instance of B. The corresponding mapping is called a *reduction*. Intuitively, problem A cannot be harder than problem B. Loosely speaking, this also means that while there are instances of problem B which can be solved efficiently, problem B is in general capable of capturing the complexity of the entire class $\mathcal{NP}$. Note that we have already implicitly applied reductions in the introduction. The approach to bypass having to solve each set problem in items 1–4 individually by modeling them as a WEIGHTED MAXIMUM SATISFIABILITY problems, requires to construct a reduction between the respective problems.

Hardness and Intractability Besides the categorization and simulation of problems, the concept of $\mathcal{NP}$-completeness also has a large impact on the existence of efficient algorithms for problems in $\mathcal{NP}$. By design of $\mathcal{NP}$-completeness, the existence of a polynomial-time algorithm for a *single* $\mathcal{NP}$-complete problem implies the existence of a polynomial-time algorithm for *each* problem in $\mathcal{NP}$. The well-known decision problems which can be derived from all the optimization problems outlined in the introduction have been categorized as $\mathcal{NP}$-complete. Up to the point of publication of this thesis, there does not exist a polynomial-time algorithm for any $\mathcal{NP}$-complete problem. The general belief is that there is little hope for efficient algorithms for $\mathcal{NP}$-complete problems. $\mathcal{NP}$-complete problems are widely regarded as hard problems and are therefore also coined *intractable* problems.

The categorization of specific problems by tractability also has the potential to unveil insights on the computational complexity of optimization problems in general. Moreover, it can help derive a deeper understanding of the involved complexity classes. For a thorough introduction to the theory of $\mathcal{NP}$-completeness and intractability in general, confer the excellent book of Garey and Johnson [40].

1.1.1 Standard Set Problems

As the examples in items 1–4 demonstrate, weighted set problems are *fundamental combinatorial optimization* problems with a wide range of applications. In general, the input to a weighted set problem consists of a *set system* along with a *weight function* on the set system. The task is to compute a solution maximizing or minimizing some objective function on the set system while obeying certain constraints. Besides the examples outlined above, the application of weighted standard set problems ranges from crew scheduling in transportation networks and machine scheduling problems to facility location problems [52].

Intractability of Weighted Standard Set Problems Now, focus on the weighted SetPacking and SetCover problem, which are well-known representatives of standard set problems. For an annotated bibliography on the application of the two problems, confer the survey by Balas and Padberg [13]. In the weighted SetPacking problem, we are given a set $\mathcal{C}$ of weighted sets over a finite set $\mathcal{B}$. The objective is to compute a set S of pairwise disjoint sets from $\mathcal{C}$, which maximizes the sum of the weights of the sets in S. Analogously, on an identical input, in the SetCover problem the objective is to compute a covering S of $\mathcal{B}$—a subset S of $\mathcal{C}$ in which every element from $\mathcal{B}$ is contained in at least one set—which minimizes the sum of the weights of the sets in S. The book of Garey and Johnson [40, page 221ff.] pools a total of eleven prominent unweighted set problems as decision problems in [SP1]–[SP11]. The standard set problems listed there range from matching and packing to covering problems, including unweighted decision problems which can be derived from the set problems outlined in items 1–4. For all set problems listed in [SP1]–[SP11], deciding if a solution exists is $\mathcal{NP}$-complete.

1.1.2 Satisfiability Problems

The Satisfiability problem and its extensions are well-studied fundamental problems in computer science whose groundbreaking results lead to the development of entire subdisciplines.

The Satisfiability Problems In the optimization problem Weighted Maximum Satisfiability, an instance consists of a finite set of weighted *clauses* $\mathcal{C}$ over a finite set of *variables* $\mathcal{X}$. A clause $C_i \in \mathcal{C}$ is a set of *literals*, where a literal is a variable or its negation. An *assignment* for the set of variables $\mathcal{X}$ is a function $\mathcal{X} \to \{0, 1\}$. For a given assignment, a literal flips the value of its variable if the literal is the negation of its variable, otherwise it inherits the assignment of its variable. A clause is *satisfied* by a given assignment if at least one of its literals is assigned 1. Note that this corresponds to connecting the literals by the boolean operator `OR`. The task is to compute an assignment maximizing the sum of the weights of the satisfied clauses. The Satisfiability problem denotes a decision problem derived from Weighted Maximum Satisfiability where each clause has weight one. The task is now to decide if there exists an assignment which satisfies *all* clauses simultaneously.

Theoretical and Practical Impact In complexity theory, the Satisfiability problem was the first decision problem shown to capture the complexity of the entire class $\mathcal{NP}$ [23]; Satisfiability was the *initial* $\mathcal{NP}$*-complete problem.* The intractability of Satisfiability led to the $\mathcal{NP}$-completeness of 21 combinatorial and graph theoretical problems, now known as "Karp's 21 $\mathcal{NP}$-complete Problems" [60]. Succeedingly, numerous other problems were proven to be $\mathcal{NP}$-complete and the theory of completeness was carried beyond decision problems; confer e.g. Ausiello et al. [11]. Extensions of the Satisfiability problem are also fundamental for other complexity classes, such as the class of decision problems which can be solved by a polynomial-*space* bounded Turing machine [102]; the corresponding class is called $\mathcal{PSPACE}$.

The $\mathcal{NP}$-completeness of Satisfiability does not only show the intractability of the problem itself but can also serve as a starting point for solving other intractable problems. By definition of $\mathcal{NP}$-completeness, the Satisfiability problem is able to simulate each problem in $\mathcal{NP}$. Furthermore, many problems can be modeled as Satisfiability problems. This approach is used to solve problems arising from bounded model checking, hardware design, planning, software verification, and countless other problems from various disciplines [14]. The constant demand for solving the seemingly omnipresent Satisfiability problem and its extension in practice, led to the development of numerous *solvers.* For a survey of the applications of Satisfiability, its derivatives, and solvers confer Gu et al. [46] and the book of Biere et al. [14].

1.1.3 Approximation of Intractable Problems

The decision problems which can be derived from the weighted set problems in the introduction and also the Satisfiability problem are known to be $\mathcal{NP}$-complete, as outlined in the previous paragraphs. These problems are in good company, as over 40 years of research in computer science unveiled that roughly 3,000 problems, and counting, are intractable. Garey and Johnson [40] compiled a list of 300 fundamental $\mathcal{NP}$-complete problems and included this catalog in their book. The problems which are identified as $\mathcal{NP}$-complete are on the one hand of great interest but on the other hand have resisted tractability up to now. Therefore, numerous approaches have been developed to tackle these problems.

Approximation Algorithms The general idea of approximation algorithms is to sacrifice solution quality in return for a polynomial running time. Hence, instead of striving for (globally) optimal solutions, one now pursues *approximate solutions*, i. e. solutions which might not be (globally) optimal, but are at least *good*, with respect to some measure or the personal perspective. For instance, approximation algorithms compute solutions whose objective function value is guaranteed not to be more than a predetermined factor away from an optimum. Several approaches have been developed for SetPacking, SetCover, HittingSet, and other weighted standard set problems. For a survey of approximation algorithms for covering and packing problems, confer Paschos [88], Hoffman and Padberg [52] (and references therein). The Satisfiability problem and its extensions have been intensively studied in the literature with numerous approaches being developed for these fundamental problems. For a survey of the various approaches, we refer the reader to the surveys by Gu [45], Gu et al. [46], Kautz and Selman [61], the corresponding chapters in the handbooks by Gonzalez [43], Kao [58], and the recently published book by Biere et al. [14]. For a general overview of approximation algorithms, confer the book of Hochbaum [51].

Metaheuristics Unfortunately, some problems in $\mathcal{NP}$ even *resist polynomial-time approximation* in the sense that the existence of a polynomial-time algorithm which computes an approximate solution would directly imply that $\mathcal{P} = \mathcal{NP}$. For example, unless $\mathcal{P} = \mathcal{NP}$, SetPacking does not admit a constant-ratio polynomial-time approximation algorithm [9]; moreover, SetCover and HittingSet do not admit a polynomial-time approximation scheme [40, 84] (an approximation algorithm with an additional input parameter $\epsilon > 0$, which—when neglecting ϵ—produces a solution in polynomial time that is within a factor of $1 + \epsilon$ from being optimal), unless $\mathcal{P} = \mathcal{NP}$. Håstad [49] shows that for Weighted Maximum Satisfiability where all clauses have three literals and all weights are equal to one, no approximation algorithm can achieve a performance guarantee better than 7/8, unless $\mathcal{P} = \mathcal{NP}$.

A popular approach to tackle problems which even resist polynomial-time approximation at a certain degree, are *(meta-)heuristics.* Nearly all metaheuristics like local search, simulated annealing, evolutionary algorithms, to name a few popular ones, share the same general approach. Starting from a given (population of) solution(s), they compute a new (population of) solution(s) and continue the computation with the new one(s); here, solutions with a higher quality, with respect to the objective function, are preferred. For the local search approach, the preference for improved solutions is strict, i. e. the computation only proceeds with strictly better solutions. This approach is most suitable for convex optimization problems since in convex optimization problems, local optima coincide with global optima. Metaheuristics with strict preferences for improved solutions are tailored to find local optima, by design. If local optima are frequent in the solution space but of rather different quality with respect to the objective function, then local search with a strict preference for new solutions might be regarded as a drawback. For this, metaheuristics with non-strict preferences for new solutions are suitable, since they can escape local optima. However, the focus in this thesis is on the local search approach and in particular the complexity of computing a local optimum.

Local search is one of the most frequently used approaches to solve hard combinatorial optimization problems. Famous examples of successful applications of local search algorithms are the SIMPLEX method for solving linear programs [15, 103], the k-OPT heuristic for finding solutions of the TRAVELING SALESMAN PROBLEM [1], and the k-MEANS algorithm for clustering objects [26, 54]. For further information on local search, its complexity, and related problems confer also Solis-Oba [100], Yannakakis [108, 110].

1.2 Local Search

Local search is a standard approach to approximate solutions of hard combinatorial optimization problems. In this section, we first present the concept of local search in Subsection 1.2.1, before surveying successful applications of local search in Subsection 1.2.2. We present the class $\mathcal{PLS}$, introduced by Johnson, Papadimtriou, and Yannakakis [56], as a theoretical framework to investigate the complexity of local search in Subsection 1.2.3. Subsection 1.2.4 then draws the connection between local search and game theory. Let us stress that in this thesis, we merely focus on the complexity of computing locally optimal solutions. For an overview of the quality of solutions obtained via local search, confer the survey by Angel [8].

1.2.1 The Concept of Local Search

Local search is a natural and intuitive approach to compute locally optimal solutions: Starting from an arbitrary feasible solution, a sequence of feasible solutions is iteratively generated, such that each solution is contained in the predefined *neighborhood* of its predecessor solution and strictly improves a given *cost function.* If no improvement within the neighborhood of a solution is possible, a *local optimum* (or *locally optimal solution*) is found. Hence, a local search problem may be constructed from an arbitrary optimization problem by superimposing a neighborhood structure on the set of feasible solutions. A generic local search algorithm for computing locally optimal solutions may now work as given in Algorithm 1.

For an instance I of a local search problem P, an *improving step* is a move from a solution s of I to a better neighboring solution s' of I; note that this corresponds to a

Algorithm 1 GenericLocalSearch (Instance I of Local Search Problem P)

```
1: Compute an initial feasible solution $\mathsf{s}$ for instance $I$
2: while $\mathsf{s}$ is not locally optimal do
3:     Compute a better solution $\mathsf{s}'$ within the neighborhood of $\mathsf{s}$
4:     Set $\mathsf{s} := \mathsf{s}'$
5: end while
6: return $\mathsf{s}$
```

call of the **while** loop in lines 2–5 in Algorithm 1. Now, assume that each single step in lines 1–5, including verifying or disproving local optimality of a solution s in line 2 and the implicit computation of the cost of a solution s in line 3 is polynomial-time computable. Then, there is no a priori guarantee that the number of calls of the **while** loop in lines 2–5 and hence the length of every sequence of improving steps is bounded by a polynomial in the size of the input. Now, consider the following simple minimization local search problem:

Definition 1.1 (Exp [78])**.** *Instances of* Exponential *(in short* Exp*) are natural numbers $n \in \mathbb{N}$, encoded as a binary string. The set of feasible solutions consists of all binary strings $\{0,1\}^n$. The cost of a solution is the natural number the binary string represents and the single neighbor is the natural number the binary string represents minus one; the neighborhood of the all-zero vector is empty.*

Starting with the all-one vector, it is obvious to see that by design, every sequence of improving steps in a given instance of Exponential requires exponentially many steps to reach the locally optimal all-zero vector, which is also the single local optimum. This shows that *there exist local search problems and initial feasible solutions such that every sequence of improving steps has exponential length.* Despite this general negative result on the rapid convergence of sequences of improving steps, local search algorithms often require only a few steps to compute a locally optimal solution in practice. However, the running time is in many times pseudo-polynomial and even exponential in the worst case, as pointed out above. The concept of local search reaches back to the late 1950s and early 1960s when it was first applied to the Traveling Salesman Problem [71]. Since then it has been successfully applied to a wide range of problems from different areas.

1.2.2 Successful Applications of Local Search

Here, we consider three combinatorial optimization problems in which local search celebrated its greatest practical successes, namely LinearPrograms, the Traveling Salesman Problem, and the problem of Clustering objects.

LinearPrograms An instance of a LinearProgram consists of a matrix A and two vectors b and c. The task is to compute a vector x maximizing $c^T x$ such that $Ax \leq b$. Due to the convexity of LinearPrograms, local optima coincide with global optima.

This suggests the use of local search in a natural way. In 1947, Danzig [24] introduced the famous SIMPLEX Algorithm which computes an optimum by starting at a vertex of the polytope formed by the constraints $Ax \leq b$ and iteratively advances to better vertices, with respect to $c^T x$, until an optimum is reached. Since its introduction, the SIMPLEX Algorithm has been successfully applied to LINEARPROGRAMs originating from a wide range of applications, including scheduling problems, production planning, routing problems, and problems arising in game theory. Although the running time of the SIMPLEX algorithm was quickly observed to be polynomial on "real-world" instances, significant progress in speeding up the algorithm took until the end of the 1980s [15].

The existence of exponentially long improving sequences caught the interest in the early years of the SIMPLEX method. LINEARPROGRAMs were constructed on which the SIMPLEX method with the steepest descent pivoting rule takes an exponential number of pivoting steps [65]. Similar results were shown for other pivot rules [64, 103]. Despite that, Kalai and Kleitman [57] showed that for every initial solution of a LINEARPROGRAM with n inequalities and d variables, the distance to a local optimum in the polytope formed by the inequalities is at most $n^{\log d+2}$ pivot steps. This raises the question whether it is possible to find such a path efficiently. However, computing an optimum point of a LINEARPROGRAM is known to be in $\mathcal{P}$ due to the Ellipsoid method by Khachiyan [62] which uses an approach different from local search. Karmarkar [59] subsequently introduced the interior point method which also requires polynomial time and even outperforms the SIMPLEX algorithm in some practical applications.

The Traveling Salesman Problem An instance of the TRAVELING SALESMAN PROBLEM consists of a complete undirected edge-weighted graph. The task is to compute a round trip (or tour) of minimum weight which visits each node of the given graph exactly once. Already in the 1960's, local search was proposed to solve the TRAVELING SALESMAN PROBLEM in practice [71]. The probably most frequently used local search heuristic for the TRAVELING SALESMAN PROBLEM is the k-OPT heuristic, where $k = 2$ in most cases. Starting from an initial tour, up to k edges may be rearranged in an improving step in order to construct a tour of lower cost [71]. An alternative heuristic, also based on local search, is the more complex LIN-KERNIGHAN heuristic (confer Johnson and McGeoch [55] for a formal definition) [72]. For random and "real-world" Euclidean instances, the 2-OPT and LIN-KERNIGHAN heuristics are known to compute very good tours within a sub-quadratic number of improving steps [55, 91].

Despite the fast convergence on practical instances, it was shown that there exist instances and initial solutions of the TRAVELING SALESMAN PROBLEM for which the k-OPT heuristic for $k \geq 2$ can take exponentially many improving steps [19, 73]. However, these instances do not satisfy the triangle inequality and the existence of such instances in the metric TRAVELING SALESMAN PROBLEM remained unsettled for a long time. Eventually, Englert et al. [33] prove the existence of Euclidean instances on which the 2-Opt heuristic can take exponentially many improving steps.

Clustering The problem Clustering asks for a partition of a set of data into subsets (also known as clusters) such that some given measure for the similarity within the clusters is maximized. The problem of clustering data occurs in many applications, including pattern recognition, data compression, and load balancing with problem-specific nuances, depending on the application. A well-studied algorithm for clustering points in the Euclidean space is the k-Means algorithm. It starts with an initial set of k centers for the clusters; each data point is assigned to its closest center. Then, the solution is improved by repeatedly performing the following two steps: First, a new center is defined for each cluster as the average of all points of the cluster, i. e. the "mean" of all points in the specific cluster. Finally, all points are assigned to their clusters, according to the minimum distance of the new center points. Note that in each improving step, the sum of the distances of the data points to their corresponding closest center, decreases; this can be regarded as a potential function. Let us remark that in the Euclidean space, an improving step of the k-Means algorithm is uniquely determined.

For practical instances, the number of steps of the k-Means algorithm was observed to be linear in the number of data points [26]. Similar to the two previously mentioned famous problems, there exist instances and initial solutions of the Clustering problem for which the k-Means algorithm takes an exponential number of improving steps to converge [105].

Randomized Instances and Smoothed Complexity For many years, the running time of local search algorithms in general, and the Simplex method in particular, was observed to be very low on most instances occurring in practical applications. Inspired by this observation, the complexity of the Simplex algorithm was investigated for numerous distributions of random inputs and shown to be in expected polynomial time [6, 16, 99]. The same observation was made for the 2-Opt heuristic for computing solutions of the Traveling Salesman Problem on random instances in the unit hypercube [19]. However, in contrast to the constructed inputs for which an exponential number of improving steps are possible, it can be argued that the random instances may have certain properties that do not reflect the properties of real-world instances.

In order to understand why the running time is polynomial on so many real-world instances, Spielman and Teng [101] introduced the notion of *smoothed complexity*. In a nutshell, smoothed complexity measures the expected running time of an algorithm under small random perturbations of the input. The authors show that the Simplex algorithm for LinearPrograms has polynomial smoothed complexity. Subsequently, the notion of smoothed complexity was adapted for algorithms of various problems, including other local search algorithms. The smoothed complexity of the 2-Opt heuristic for Euclidean instances was shown to be polynomial [33]. Recently, the k-Means algorithm was also shown to have polynomial smoothed complexity [10].

1.2.3 A Framework to Investigate the Complexity of Local Search

Subsection 1.2.1 outlines the concept of local search and shows that in general, there does not exist an upper bound on the length of every sequence of improving steps which is polynomial in the size of the input. However, the previous subsection outlined that in practice, local search algorithms often require only a few steps to compute a locally optimal solution.

Formalizing Local Search In order to investigate the computational complexity of local search algorithms, Johnson, Papadimtriou, and Yannakakis [56] introduced the class $\mathcal{PLS}$ (short for "Polynomial-time Local Search") in 1988. Essentially, a problem in $\mathcal{PLS}$ is given by some *minimization* or *maximization problem* over instances with finite sets of *feasible solutions* along with a non-negative integer *cost function.* A *neighborhood structure* is superimposed over the set of feasible solutions, with the property that a local improvement in the neighborhood can be found in polynomial time. Additionally, an initial solution and the cost of a solution can be computed in polynomial time for each instance of a $\mathcal{PLS}$ problem. The objective is to find a *locally optimal* solution, i. e. a solution which lacks a better neighboring solution. Note that this description matches the definition of a local search problem in Subsection 1.2.1, except for the restrictions of a polynomial bound on the running time of the corresponding algorithms.

Identifying and Categorizing Local Search Problems In order to establish relationships between $\mathcal{PLS}$ problems and to further classify them, Johnson, Papadimtriou, and Yannakakis [56] define the notion of a *$\mathcal{PLS}$ reduction.* In essence, a $\mathcal{PLS}$ reduction from $\mathcal{PLS}$ problem A to a $\mathcal{PLS}$ problem B maps instances of A to instances of B in polynomial time such that local optima of B can be transformed into local optima of A in polynomial time. Intuitively, problem A cannot be harder than problem B. A $\mathcal{PLS}$ problem B is *$\mathcal{PLS}$-complete* if every problem in $\mathcal{PLS}$ is $\mathcal{PLS}$-reducible to B. The concept of $\mathcal{PLS}$-completeness is in line with the general use of completeness in complexity theory. In the case of $\mathcal{PLS}$, the concept entails that if *some* $\mathcal{PLS}$-complete problem is polynomial-time solvable, then *all* problems in the class $\mathcal{PLS}$ are polynomial-time solvable. Similar to $\mathcal{NP}$-complete problems, no polynomial-time algorithm is known for any $\mathcal{PLS}$-complete problem. $\mathcal{PLS}$-complete problems are widely regarded as hard problems and are therefore also referred to as *intractable* problems.

A drawback of $\mathcal{PLS}$ reductions from A to B, as outlined above, is that they may introduce additional sequences of improving steps in instances of B such that shortcuts between two solutions of an instance of A are opened up, under some reduction. Hence, there is no a priori guarantee that the exclusiveness of exponentially long sequences of improving steps is preserved under this type of reduction. Tight reductions circumvent this obstacle, by enforcing a more structured reduction. In a nutshell, a $\mathcal{PLS}$ reduction is *tight* if all sequences of improving steps in the instance created by the reduction correspond to sequences of improving steps in the original problem whose length may only be increased by introducing intermediate solutions. We present formal definitions for all of the above concepts in Chapter 2. Overall, not many problems are known to be $\mathcal{PLS}$-complete, since reductions are mostly technically involved, which seems to be in large parts due to the transformation of the neighborhood under the reduction.

Local Search in Complexity Theory In general, not much is known about the relation of $\mathcal{PLS}$ to other complexity classes. By construction, $\mathcal{PLS}$ is contained in $\mathcal{TFNP}$, the class of total functions from $\mathcal{NP}$. It is though rather unlikely that $\mathcal{PLS}$ equals $\mathcal{TFNP}$, since this would imply that $\mathcal{NP}$ is closed under complement [56]. Nevertheless, some notions of hardness hold for $\mathcal{PLS}$-complete problems, to which Circuit/Flip (confer Definition 2.12 in Chapter 2 for a formal definition) is reducible via a sequence of tight reductions [95]. The *standard algorithm problem* is, for a given instance I of a $\mathcal{PLS}$ problem L and some feasible solution s, to compute a local optimum $\mathsf{s}^\star$ which is reachable from s by successive improvements. The standard algorithm problem of Circuit/Flip is known to be $\mathcal{PSPACE}$-complete [87, 109]. Tight reductions as sketched above preserve the $\mathcal{PSPACE}$-completeness of the standard algorithm problem [87, 109]. Moreover, tight reductions also preserve the property that there exist instances and initial feasible solutions such that every sequence of improving steps has exponential length before reaching a locally optimal solution [56]. Though a handful of problems have been shown to be $\mathcal{PLS}$-complete, the knowledge about $\mathcal{PLS}$ is still very limited and not at all comparable with our rich knowledge about $\mathcal{NP}$.

1.2.4 Local Search and Game Theory: Optimization in Competition

Recently, the field of local search has attracted additional attention from *game theory*, considering the *complexity of computing a pure Nash equilibrium.* In this subsection, we only consider pure Nash equilibria; thus, we omit pure for sake of readability.

Non-Cooperative Games and Nash Equilibria In game theory, a *game* is characterized by a set of *players*, a set of *strategies* for the players, and a set of *cost functions* for the players. Here, the cost function of each player represents his private cost and depends on the choices of strategies of all other players. A *state of the game* is an assignment of each player to a strategy. A famous solution concept for non-cooperative games is the *Nash equilibrium* [80]. In a Nash equilibrium, no player can unilaterally deviate and strictly improve his private cost. Hence, a Nash equilibrium defines a *stable state of the game.* The definition of a Nash equilibrium gives rise to a simple dynamics, known as *Nash dynamics*, to compute these stable states: In each round, a single player is allowed to perform a *selfish step*, i. e. unilaterally change his strategy and strictly improve his private cost; the dynamics terminates once no such player exists and is synonymously referred to as the SelfishSteps algorithm. In the special Nash dynamics known as *best-response dynamics*, each player selects a strategy which maximizes the improvement of his private cost.

Nash Equilibria and Local Search Problems Proving the existence of Nash equilibria for classes of games is usually done using some *potential function* argument. A potential function maps every state of the game to a positive integer such that every selfish step of each player strictly decreases the potential function. Local optima of potential functions then coincide with *Nash equilibria.* In case of polynomial-time computable potential functions, the problem of finding a Nash equilibrium can be formulated as a $\mathcal{PLS}$ problem, where the neighborhood structure is superimposed by the Nash dynamics. For a thorough introduction to game theory, confer the book of Myerson [79].

1.2.5 The Class of Congestion Games

Inspired by road traffic and more recently the internet, the class of congestion games has been under severe scrutiny for the last years. Congestion games are a widely accepted model to investigate the behavior and performance of large-scale distributed networks with autonomous participants.

Congestion Games and Nash Equilibria Congestion games are non-cooperative games where n weighted *players* myopically select *strategies* s_i from their set of strategies S_i (subsets of the set of shared resources $\mathcal{R}$) such that their *individual delay* on all resources in the current state of the game $\mathsf{s} = (s_1, \dots, s_n)$ is minimized. Here, the individual delay of a player is the sum of the delays on each resource the player is using. The delay on resource $r \in \mathcal{R}$ is the value of the delay function $d_r : \mathbb{N} \to \mathbb{N}_0$ for the sum of the weights of the players using the resource. A congestion game is *unweighted*, if the weights of all players are equal. In that case, $n_r(\mathsf{s})$ denotes the number of players using resource $r \in \mathcal{R}$ in state s of the game. In *network congestion games*, the set of strategies for each player corresponds to all his source-sink paths. A congestion game is *symmetric* if all players have the same set of strategies and *asymmetric* otherwise.

Unweighted congestion games are known to possess an exact potential function $\Phi(\mathsf{s}) = \sum_{r \in \mathcal{R}} \sum_{i=1}^{n_r(\mathsf{s})} d_r(i)$, also known as Rosenthal's potential function [92]; hence, computing a Nash equilibrium can be formulated as finding a minimum of the potential function. For weighted congestion games, a recent result [48] shows that affine linear [37] and exponential [83] delay functions are the only general classes which admit Nash equilibria. Note that for unweighted congestion games, the Nash dynamics converges after at most a polynomial number of iterations, if all delay functions are polynomials. In contrast to that, if a potential function exists for a weighted congestion game, then its number of states is no longer polynomially bounded in general.

Restricted Network Congestion Games In this thesis, our focus in the area of game theory is on the class of restricted network congestion games. Restricted network congestion games are network congestion games where *for each unweighted player* there *exits a set of edges* which he is *not allowed to use.* Rosenthal's potential function [92] guarantees the existence of a Nash equilibrium. As in unweighted congestion games, the problem of computing a Nash equilibrium can be formulated as a $\mathcal{PLS}$ problem.

1.3 Central Questions of this Thesis

To the best of our knowledge, the exact tractability of computing locally optimal solutions of weighted standard set problems, of Weighted Maximum Satisfiability problems, and the tractability of computing Nash equilibria in restricted network congestion games is unsettled. This leads us to the below stated central questions which we investigate in this thesis. We classify these central questions as superior questions which affect all problems in the class $\mathcal{PLS}$ and questions specific to the problems we consider. Superior questions are stated in Subsection 1.3.1, problem-specific questions are listed in Subsections 1.3.2–1.3.4.

1.3.1 Superior Questions

The Algorithm 1 GenericLocalSearch presented in Subsection 1.2.1 succeeds in computing some locally optimal solution but lacks a general polynomial-time bound on its running time. The most evident and superior question is thus whether there exist potentially sophisticated algorithms for the problems we investigate which compute locally optimal solutions in polynomial time.

Question 1. *What is the complexity of computing a locally optimal solution of the problems we consider?*

In this thesis, our emphasis will not be on particular algorithms, but rather on *demarcating the tractability* of computing locally optimal solutions of the problems we study. For our investigation, we use the framework of $\mathcal{PLS}$, as introduced by Johnson, Papadimtriou, and Yannakakis [56]. We try to either present polynomial-time algorithms which compute locally optimal solutions or show the intractability of the respective problems, using reductions within the framework of $\mathcal{PLS}$. Intractability results will then suggest that there is little hope for polynomial-time algorithms for the corresponding problems. Most of the reductions we present will be rather technically involved and consist of problem-specific constructions. Nonetheless, we are interested in *commonalities* between the reductions we present. These commonalities might lead the way to uncover some common proof patterns or schemes behind our reductions. Ideally, we would like to have something similar to a "recipe" for future $\mathcal{PLS}$ reductions.

Question 2. *Is it possible to identify a general structure behind the reductions we present?*

A $\mathcal{PLS}$-completeness proof not only shows the intractability of the corresponding problem, but also provides a valuable insight perspective of the problem. Additionally, the $\mathcal{PLS}$-completeness proofs we present might provide some feedback for a better understanding of hard problems in the class $\mathcal{PLS}$ and how to identify them according to tractability.

Question 3. *What are the potential sources of intractability in the problems we show $\mathcal{PLS}$-completeness for?*

A potential answer to the latter question might prove useful in identifying if a future $\mathcal{PLS}$ problem tends to be tractable or resists tractability. In the long run, this might fuel the hope for a non-complex, general criterion for $\mathcal{PLS}$-hardness.

1.3.2 Restricted Network Congestion Games

Considering congestion games, we are interested in the complexity of computing Nash equilibria when either the number of players, resources, or the size of each strategy is a priori fixed. For a constant size of each strategy, Fabrikant et al. [34] give a surrounding intractability answer, requiring only two strategies per player which can be further lowered to at most five resources per strategy [34, 104]. We present a more comprehensive overview of these and other results addressing congestion games in Subsection 3.2.1. For a fixed number of resources, computing a Nash equilibrium is polynomial-time solvable since the total number of strategies is bounded by a polynomial in the size of the input [4]. Hence, the tractability of computing Nash equilibria in congestion games in the presence of an a priori fixed number of players remains unsettled.

Question 4. *Which impact does the number of players have on the complexity of computing a Nash equilibrium in a given congestion game?*

For our investigation, we consider the subclass of restricted network congestion games (confer Definition 2.13 in Chapter 2 for a formal definition). Let us remark that Question 4 is also addressed in the work of Ackermann and Skopalik [4]. There, the authors show that computing a Nash equilibrium in a restricted network congestion game involving *three players* only is $\mathcal{PLS}$-complete. In this thesis, we are interested in the tractability of computing Nash equilibria in restricted network congestion games involving two players only.

Question 5. *What is the complexity of computing a Nash equilibrium in a restricted network congestion game involving two players only?*

Let us stress that an answer to Question 5 settles the complexity of computing Nash equilibria in restricted network congestion games, since for one player, computing a Nash equilibrium reduces to finding a shortest path, which is polynomial-time computable.

1.3.3 Maximum Constraint Assignment

Regarding the optimization problem Weighted Maximum Satisfiability, we study the tractability of computing locally optimal solutions of the Maximum Constraint Assignment problem (in short MCA; confer Definition 2.15 in Chapter 2 for a formal definition). In a nutshell, the MCA problem is a local search version of weighted Generalized Maximum Satisfiability (confer problem [LO6] in the book of Garey and Johnson [40] for a formal description) on constraints (functions mapping assignments to positive integers) over variables with higher valence. Additional parameters in (p, q, r)-MCA simultaneously limit the maximum length p of each constraint, the maximum appearance q of each variable and its valence r. For our investigation, we build on a work from Krentel [67], who outlines that $(4, 3, 3)$-MCA is $\mathcal{PLS}$-complete. We will provide more information on the work of Krentel [67] in Subsection 3.1.4 and Section 6.2. We are mostly interested in simultaneous small combinations of both the maximum length of each constraint and the maximum appearance of each variable. Furthermore, we are also interested in the special case of binary variables. As we believe that the Maximum Constraint Assignment problem is a fundamental problem for the class $\mathcal{PLS}$, we would like to delimit the intractability of computing locally optimal solutions of the Maximum Constraint Assignment problem.

Question 6. *What are the bounds on the intractability of the* Maximum Constraint Assignment *problem?*

Notably, $(2, 2, r)$-MCA is solvable in polynomial time for every $r \in \mathbb{N}$ and this might additionally fuel the hope for demarcating results.

1.3.4 Weighted Standard Set Problems

Local search algorithms are widely used in practice with large success to approximate weighted standard set problems such as SetPacking and SetCover, presented in Subsection 1.1.1. For our investigations, we choose local search versions of problems [SP1]–[SP10] in the book of Garey and Johnson [40, page 221ff.] as representatives of a broader class of standard set problems. We are again interested in demarcating the tractability of computing locally optimal solutions of the corresponding weighted standard set problems (confer Definitions 2.20–2.29 in Chapter 2 for a formal definition). Tight $\mathcal{PLS}$ reductions then provide a feedback for existing heuristics in the sense that there are instances and initial feasible solutions which require an exponential number of improving steps before reaching a locally optimal solution. When focusing on hardness results, we are also interested in the structure of intractable instances of the problems we investigate.

Question 7. *Does the approach of local search yield polynomial-time algorithms for the weighted standard set problems we consider?*

Let us stress once more that in our investigations, we merely focus on the complexity of computing a locally optimal solution, regardless of the solution quality. When presenting polynomial-time algorithms we therefore aim to reduce the degree of complexity of the algorithms; we suggest to regard them as proof-of-concept algorithms, rather than algorithms to directly apply. To the best of our knowledge, this is one of the first investigations of weighted standard set problems in the context of $\mathcal{PLS}$.

1.4 Publications

This thesis is founded on four publications [28, 29, 31, 78] which are joint work with the respective co-authors and are incorporated as follows: Predominantly Chapter 3, but also parts of Chapter 1 and 8 are founded on a publication by Monien, Dumrauf, and Tscheuschner [78] in the Proceedings of the 37th International Colloquium on Automata, Languages and Programming (ICALP 2010). Chapter 5 is based on a paper by Dumrauf and Monien [28] submitted to publication in Parallel Processing Letters (PPL). Chapter 6 is based on a paper by Dumrauf and Monien [31] submitted to publication in Theoretical Computer Science (TCS). Chapter 7 builds on a publication by Dumrauf and Süß [29] in the Proceedings of the 6th Conference on Computability in Europe (CiE 2010).

1.5 Roadmap of this Thesis

This thesis is organized as follows: Chapter 2 presents the notation, classes and problems used in this thesis. In Chapter 3, we survey the research on the complexity of computing locally optimal solutions in the framework of $\mathcal{PLS}$. In Chapter 4, we summarize the main results we show in this thesis, including a graphical overview (Figure 4.1) of our main $\mathcal{PLS}$ reductions which we present in this thesis. Chapter 5 investigates the complexity of computing a Nash equilibrium in a given restricted network congestion game involving two players only and discusses Questions 4 and 5. In Chapter 6, we study the complexity of computing locally optimal solutions of the MAXIMUM CONSTRAINT ASSIGNMENT problem and discuss Question 6. In Chapter 7, we investigate the complexity of computing locally optimal solutions of weighted standard set problems and discuss Question 7. We close this thesis with a general conclusion in Chapter 8, discuss Questions 1–3, state various open problems, and outline directions for further research.

Chapter 2

Notation

In this chapter, we introduce the *notation*, *classes*, and *problems* used in this thesis. Section 2.1 presents the definition of a $\mathcal{PLS}$ problem, the class $\mathcal{PLS}$, and the definition of a $\mathcal{PLS}$ reduction. In Section 2.2, we introduce the $\mathcal{PLS}$ problems considered in this thesis.

A Note on Our General Notation For sake of readability, we denote computational problems as well as algorithms in *small capitals*; sets are denoted in *calligraphic letters*, where the special sets of complexity classes have names of length at least three. For some set $\mathcal{S}$, denote $\mathcal{S}_0 := \mathcal{S} \cup \{0\}$ and $2^{\mathcal{S}}$ the power set of $\mathcal{S}$. For all $j, k \in \mathbb{N}_0$, denote $[k] := \{1, \ldots, k\}$ and

$$[j : k] := \begin{cases} \{j, \ldots, k\} & \text{if } j \leq k \\ \emptyset & \text{otherwise.} \end{cases}$$

Given a k-tuple T, let $P_i(T)$, for $i \in [k]$, denote the *projection* to the i-th coordinate. We denote an *arbitrary value* of a parameter by an *asterisk* symbol $(*)$.

2.1 $\mathcal{PLS}$, Reductions, and Completeness

The fundamental definitions of a $\mathcal{PLS}$ problem and the class $\mathcal{PLS}$ were introduced by Johnson, Papadimitriou, and Yannakakis [56]. The definition of a tight reduction was introduced by Schäffer and Yannakakis [95]. All definitions can also be found in the book of Aarts et al. [2]. In the following definitions, we assume that all elements of all occurring sets are encoded as binary strings of finite length.

Definition 2.1 ($\mathcal{PLS}$ Problems [56]). *A* $\mathcal{PLS}$ problem *L is characterized by seven parameters*

$$L = (D_L, F_L, c_L, N_L, \textsc{Init}_L, \textsc{Cost}_L, \textsc{Improve}_L).$$

The set of instances is given by a polynomial-time recognizable subset $D_L \subseteq \{0,1\}^*$. Every instance $I \in D_L$ has a set of feasible solutions $F_L(I)$, where feasible solutions $\mathsf{s} \in F_L(I)$ are strings that have length bounded by a polynomial in the length of I. Every feasible solution $\mathsf{s} \in F_L(I)$ has a non-negative integer cost $c_L(\mathsf{s}, I)$ and a neighborhood $N_L(\mathsf{s}, I) \subseteq F_L(I)$. $\textsc{Init}_L(I)$*,* $\textsc{Cost}_L(\mathsf{s}, I)$*, and* $\textsc{Improve}_L(\mathsf{s}, I)$ *are* *polynomial-time algorithms. Algorithm* $\textsc{Init}_L(I)$*, given an instance* $I \in D_L$*, computes an initial feasible solution* $\mathsf{s} \in F_L(I)$*. Algorithm* $\textsc{Cost}_L(\mathsf{s}, I)$*, given a feasible solution* $\mathsf{s} \in F_L(I)$ *and an instance* $I \in D_L$*, computes the cost of solution* s*. Algorithm* $\textsc{Improve}_L(\mathsf{s}, I)$*, given a feasible solution* $\mathsf{s} \in F_L(I)$ *and an instance* $I \in D_L$*, finds a better solution in* $N_L(\mathsf{s}, I)$ *or returns that there is no better one.*

For sake of readability, we drop the index of the problem, where it is is clear from the context. Note that locally optimal solutions can be found non-deterministically in polynomial time by first guessing a feasible solution and subsequently verifying local optimality. It is known that if a $\mathcal{PLS}$ problem L is $\mathcal{NP}$-hard, then $\mathcal{NP}$ is closed under complement [56].

Definition 2.2 (Improving Steps [56])**.** *Let* L *be a* $\mathcal{PLS}$ *problem and* $I \in D_L$ *be an instance of* L*. A move from solution* $\mathsf{s} \in F_L(I)$ *to solution* $\mathsf{s}' \in N_L(\mathsf{s}, I)$ *is an* improving step *if* $c_L(\mathsf{s}', I) > c_L(\mathsf{s}, I)$ *in case* L *is a maximization* $\mathcal{PLS}$ *problem and* $c_L(\mathsf{s}', I) < c_L(\mathsf{s}, I)$ *in case* L *is a minimization* $\mathcal{PLS}$ *problem.*

Definition 2.3 (All-Exp/Is-Exp Property [78])**.** *A* $\mathcal{PLS}$ *problem* L *has the* all-exp property *if there exists an instance* $I \in D_L$ *and an initial feasible solution* $\mathsf{s} \in F_L(I)$ *that is exponentially many improving steps away from any local optimum. A sequence of improving steps has* exponential length *if its number of improving steps is at least* $2^{\sqrt[k]{|I|}}$ *for some* $k \in \mathbb{N}$*. A* $\mathcal{PLS}$ *problem has the* is-exp property *if there exist instances such that there* is *a sequence of improving steps of exponential length.*

Note that by definition, the class all-exp is closed under polynomial reductions. Furthermore, for problems where each solution has at most one better neighboring solution, the is-exp property directly implies the all-exp property. In particular, this is the case for the Exponential problem and the Clustering problem where the neighborhood structure is superimposed by the k-Means algorithm. Remarkably, for the combinatorial optimization problems where local search was applied in practice, as outlined in Section 1.2.2, the is-exp property was shown but not the all-exp property.

Definition 2.4 (Local Optima [56])**.** *Let* L *be a* $\mathcal{PLS}$ *problem and* $I \in D_L$ *be an instance of* L*. A solution* $\mathsf{s} \in F_L(I)$ *is* locally optimal *(or a* local optimum*), if for every solution* $\mathsf{s}' \in N_L(\mathsf{s}, I)$*,* $c_L(\mathsf{s}', I) \leq c_L(\mathsf{s}, I)$ *in case* L *is a maximization* $\mathcal{PLS}$ *problem and* $c_L(\mathsf{s}', I) \geq c_L(\mathsf{s}, I)$ *in case* L *is a minimization* $\mathcal{PLS}$ *problem.*

The *standard algorithm problem* is, for a given instance I of a $\mathcal{PLS}$ problem L and some solution $\mathsf{s} \in F_L(I)$, to compute a local optimum $\mathsf{s}^\star \in F_L(I)$ which is reachable from s by successive improvements. There exists a $\mathcal{PLS}$ problem whose standard algorithm problem is $\mathcal{PSPACE}$-complete [87, 109].

Definition 2.5 (The Class $\mathcal{PLS}$ [56])**.** *A* search problem R *is given by a relation over* $\{0,1\}^* \times \{0,1\}^*$*. An algorithm "solves"* R*, when given* $I \in \{0,1\}^*$ *it computes an* $\mathsf{s} \in \{0,1\}^*$ *such that* $(I, \mathsf{s}) \in R$ *or it correctly outputs that such an* s *does not exist. Given a* $\mathcal{PLS}$ *problem* L*, let the according search problem be*

$$R_L := \{(I, \mathsf{s}) \mid I \in D_L, \mathsf{s} \in F_L(I) \textit{ is a local optimum}\}.$$

Then, the class $\mathcal{PLS}$ *is defined as*

$$\mathcal{PLS} := \{R_L \mid L \text{ is a } \mathcal{PLS} \text{ problem}\}.$$

Definition 2.6 ($\mathcal{PLS}$ Reductions [56])**.** *A $\mathcal{PLS}$ problem L_1 is* $\mathcal{PLS}$-reducible *to a $\mathcal{PLS}$ problem L_2 (written $L_1 \leq_{pls} L_2$), if there exist two polynomial-time computable functions $\Phi : D_{L_1} \to D_{L_2}$ and Ψ defined for $\{(I, s) \mid I \in D_{L_1}, s \in F_{L_2}(\Phi(I))\}$ with $\Psi(I, s) \in F_{L_1}(I)$ such that for all $I \in D_{L_1}$ and for all solutions $s \in F_{L_2}(\Phi(I))$, if $(\Phi(I), s) \in R_{L_2}$ then $(I, \Psi(I, s)) \in R_{L_1}$. A $\mathcal{PLS}$ problem L is* $\mathcal{PLS}$-complete*, if every $\mathcal{PLS}$ problem is $\mathcal{PLS}$-reducible to L.*

The concept of $\mathcal{PLS}$-completeness is in line with the general use of completeness in complexity theory. For $\mathcal{PLS}$, the concept entails that if *some* $\mathcal{PLS}$-complete problem is polynomial-time computable, then *all* problems in the class $\mathcal{PLS}$ are polynomial-time computable. Up to the point of publication of this thesis, there does not exist a polynomial-time algorithm for any $\mathcal{PLS}$-complete problem. $\mathcal{PLS}$-complete problems are widely regarded as hard problems and we therefore also refer to them as *intractable* problems. A $\mathcal{PLS}$ problem L is commonly regarded as *tractable* if there exists a polynomial-time algorithm which computes a locally optimal solution of each instance $I \in D_L$. Note that by definition of $\mathcal{PLS}$ reductions, there is not necessarily an a priori relation between each sequence of improving steps in $\Phi(I)$ and in I. Tight $\mathcal{PLS}$ reductions are now more structured $\mathcal{PLS}$ reductions. In a nutshell, a $\mathcal{PLS}$ reduction is *tight* if all sequences of improving steps in $\Phi(I)$ correspond to sequences of improving steps in I whose length may only be increased by introducing intermediate solutions.

Definition 2.7 (Transition Graphs [95])**.** *Let L be a $\mathcal{PLS}$ problem and $I \in D_L$ be an instance of L. The* transition graph $\text{TG}(I)$ *of instance I is a directed graph with a node for each solution $s \in F_L(I)$ and an arc $s \to t$, whenever solution $t \in N_L(s, I)$ and $c_L(t, I)$ is strictly better than $c_L(s, I)$.*

Definition 2.8 (Tight $\mathcal{PLS}$ Reductions [95])**.** *A $\mathcal{PLS}$ reduction* (Φ, Ψ) *from $\mathcal{PLS}$ problem L_1 to L_2 is* tight*, if for each instance $I \in D_{L_1}$, there exists an $\mathscr{R} \subseteq F_{L_2}(\Phi(I))$ for the image instance $J = \Phi(I) \in D_{L_2}$, such that the following properties are satisfied:*

1. *$\mathscr{R}$ contains all local optima of J.*
2. *For every solution $s \in F_{L_1}(I)$, a solution $t \in \mathscr{R}$ can be constructed in polynomial time such that $\Psi(I, t) = s$.*
3. *Suppose that the transition graph of J, $\text{TG}(J)$, contains a directed path $q \to \cdots \to q'$ such that $q, q' \in \mathscr{R}$ but all internal path vertices are outside of $\mathscr{R}$, and let $p = \Psi(I, q)$ and $p' = \Psi(I, q')$ be the corresponding feasible solutions of I. Then, either $p = p'$ or $\text{TG}(I)$ contains an arc from p to p'.*

Tight reductions are of special interest, since they preserve the $\mathcal{PSPACE}$-completeness of the standard algorithm problem [87, 109], as well as the all-exp property [56]. Note that tight reductions are transitive. This allows to define the *tight $\mathcal{PLS}$-completess* of $\mathcal{PLS}$ problems recursively:

Definition 2.9 (Tight $\mathcal{PLS}$-Completeness [78]). *$\mathcal{PLS}$ problem* Circuit/Flip *is tight $\mathcal{PLS}$-complete [87]. A $\mathcal{PLS}$ problem B is* tight $\mathcal{PLS}$-complete *if there exists a tight $\mathcal{PLS}$ reduction $A \leq_{pls} B$ for some tight $\mathcal{PLS}$-complete problem A.*

We will outline the tight $\mathcal{PLS}$-completeness of Circuit/Flip in Subsection 3.1.1. Let us remark that linear programming with the Simplex neighborhood does not have the all-exp property [57] and is therefore not tight $\mathcal{PLS}$-complete.

2.2 $\mathcal{PLS}$ Problems Considered in this Thesis

We next present the $\mathcal{PLS}$ problems which we consider in this thesis. We separate the problems into basic $\mathcal{PLS}$ problems in Subsection 2.2.1 which we reduce from in later chapters, the problem of computing a Nash equilibrium in a given restricted network congestion game in Subsection 2.2.2, local search versions of the Generalized Maximum Satisfiability problem in Subsection 2.2.3, and local search versions of weighted standard set problems in Subsection 2.2.3.

On Algorithms and the Notation for Problems For all $\mathcal{PLS}$ problems L studied in this thesis, the algorithms Init_L and Cost_L are straightforward and polynomial-time computable. The size of the neighborhood is limited by a constant which is independent of the size of the input and therefore algorithm $\text{Improve}_L(\mathsf{s}, I)$ can search the neighborhood of $N_L(\mathsf{s}, I)$ in polynomial time. For sake of readability, we write limitations to a problem as a prefix and the size of the neighborhood as a suffix.

2.2.1 Basic $\mathcal{PLS}$ Problems

The intractability results we present in this thesis rely on known intractability results for the following three maximization $\mathcal{PLS}$ problems MaxCut, CNF-Satisfiability, and Circuit/Flip. Here, MaxCut is an edge-weighted local search version of the well-known $\mathcal{NP}$ problem MaxCut (confer [ND16] in the book of Garey and Johnson [40] for a formal definition).

Definition 2.10 (MaxCut [95]). *An instance $I \in D_{\text{MC}}$ of problem* MaxCut *(in short* MC*) is an undirected graph $\mathcal{G} = (V, E)$ along with a function $\mathsf{w} : E \to \mathbb{N}_0$. The set of feasible solutions $F_{\text{MC}}(I)$ consists of all partitions $\mathsf{p} : V \to \{0, 1\}$; we use the term* cut *synonymously for a partition. Given a solution $\mathsf{p} \in F_{\text{MC}}(I)$, the cost is*

$$c_{\text{MC}}(\mathsf{p}, I) := \sum\nolimits_{e=\{u,v\}\in E, \mathsf{p}(u)\neq \mathsf{p}(v)} \mathsf{w}(e);$$

edge $e = \{u, v\} \in E$ is in the cut *if $\mathsf{p}(u) \neq \mathsf{p}(v)$. The neighborhood $N_{\text{MC}}(\mathsf{p}, I)$ of a solution $\mathsf{p} \in F_{\text{MC}}(I)$ consists of all solutions $\mathsf{p}' \in F_{\text{MC}}(I)$, where p' results from p by a single node switching its partition.*

The $\mathcal{PLS}$ problem (h)-CNFSat, we reduce from in this thesis is a clause-weighted local search version of the well-known $\mathcal{NP}$ problem Satisfiability (confer [LO1] in the book of Garey and Johnson [40] for a formal definition). Here, the parameter $h \in \mathbb{N}$ denotes the a priori fixed maximum length of each clause. We drop the prefix when we refer to instances where clauses can have arbitrary length.

Definition 2.11 ((*h*)-CNFSat [67]). *An instance $I \in D_{(h)\text{-CNFSat}}$ of* (*h*)-CNF-Satisfiability *(in short* (*h*)-CNFSat*) with $h \in \mathbb{N}$ is a set of clauses $\mathcal{C} = \{C_1, \ldots, C_m\}$ over a set of binary variables $\mathcal{X} = \{x_1, \ldots, x_n\}$. Each clause $C_i(x_{i_1}, \ldots, x_{i_{h_i}}) \in \mathcal{C}$, with $i \in [m]$ and $h_i \leq h$, is a function $C_i : [2]^{h_i} \to \{0, \mathsf{w}_i\}$ with $\mathsf{w}_i \in \mathbb{N}$ of the form*

$$C_i(x_{i_1}, \ldots, x_{i_{h_i}}) := \begin{cases} \mathsf{w}_i & \text{if } \left(\sum_{j\in[h_i]}(x_{i_j} + b_{i_j} \mod 2)\right) \geq 1 \\ 0 & \text{otherwise,} \end{cases}$$

where $b_{i_j} \in \{0,1\}$ is a constant for all $j \in [h_i]$. Parameter h_i is also called length *of C_i and parameter w_i is also called* weight *of C_i. The set of feasible solutions $F_{(h)\text{-CNFSat}}(I)$ consists of all assignments $\mathsf{a} : \mathcal{X} \to \{0,1\}$. The cost of a solution $\mathsf{a} \in F_{(h)\text{-CNFSat}}(I)$ is*

$$c_{(h)\text{-CNFSat}}(\mathsf{a}, I) := \sum\nolimits_{C_i \in \mathcal{C}} C_i(\mathsf{a}(x_{i_1}), \ldots, \mathsf{a}(x_{i_{h_i}}));$$

clause C_i is satisfied *if $C_i(\mathsf{a}(x_{i_1}), \ldots, \mathsf{a}(x_{i_{h_i}})) > 0$. The neighborhood $N_{(h)\text{-CNFSat}}(\mathsf{a}, I)$ of a solution $\mathsf{a} \in F_{(h)\text{-CNFSat}}(I)$ consists of all solutions $\mathsf{a}' \in F_{(h)\text{-CNFSat}}(I)$, where a' results from a by switching the assignment of a single variable.*

Let us remark that the notation for (*h*)-CNFSat in the above definition slightly differs from the standard notation for the $\mathcal{NP}$ problem Satisfiability. We choose this definition, since it is in line with the notation used in the Maximum Constraint Assignment problem (see Definition 2.15). Recall that the standard definition of an instance of the decision problem Satisfiability with maximum clause length $h \in \mathbb{N}$ is a set of variables $\mathcal{X} = \{x_1, \ldots, x_n\}$, a set of literals $\mathcal{L} := \{x_1, \ldots, x_n, \bar{x}_1, \ldots, \bar{x}_n\}$, and a set of clauses $\mathcal{C} = \{C_1, \ldots, C_m\}$ with $C_i = \{l_{i_1} \vee \cdots \vee l_{i_{h_i}}\}$ where $i \in [m]$, $l_{i_j} \in \mathcal{L}$, and $h_i \leq h$ such that there exists an assignment $\mathsf{a} : \mathcal{X} \to \{0,1\}$ with $f(\mathsf{a}) = 1$, where $f(\mathsf{a}) = \bigwedge_{i=1}^{m}(\mathsf{a}(l_{i_1}) \vee \cdots \vee \mathsf{a}(l_{i_{h_i}}))$. Note that in our definition of (*h*)-CNFSat, the *constants* b_{i_j} take the role of *negating variables.*

Circuit/Flip denotes the $\mathcal{PLS}$ problem of finding an input assignment for a given feedback-free boolean circuit $\mathcal{S}$ where the output, treated as a binary number, cannot be increased by flipping a single input bit.

Definition 2.12 (Circuit/Flip [56]). *An instance $I \in D_{\mathrm{C/F}}$ of problem* Circuit/Flip *(in short* C/F*) is a feedback-free boolean* circuit *$\mathcal{S}$ consisting of a set of* gates *$\mathcal{G}$ and a set of* links *$\mathcal{L}$; denote* input links *$X_1, \ldots, X_n$ and* output links *$Z_1, \ldots, Z_n$. The circuit $\mathcal{S}$ defines a function $\mathsf{R} : \mathcal{L} \times \{0,1\}^n \to \{0,1\}$. Here, $\mathsf{R}(\ell, (x_1, \ldots, x_n))$ is the value computed by $\mathcal{S}$ for link ℓ, if input links $X_1, \ldots, X_n$ have input bits $x_1, \ldots, x_n$. For output link Z_i with $i \in [n]$ and input bits $\mathsf{x} = (x_1, \ldots, x_n)$, denote $Z(\mathsf{x})_i := \mathsf{R}(Z_i, \mathsf{x})$. The set of feasible solutions of instance I is $F_{\mathrm{C/F}}(I) :=$*

$\{0,1\}^n$. *The cost of a solution* $\mathsf{x} = (x_1, \dots, x_n) \in F_{\mathrm{C/F}}(I)$ *is defined as*

$$c_{\mathrm{C/F}}(\mathsf{x}, I) := \sum\nolimits_{i=1}^{n} 2^{n-i} Z(\mathsf{x})_i.$$

The neighborhood $N_{\mathrm{C/F}}(\mathsf{x}, I)$ *of a solution* $\mathsf{x} \in F_{\mathrm{C/F}}(I)$ *consists of all solutions* $\mathsf{x}' \in F_{\mathrm{C/F}}(I)$*, where* x' *results from* x *by flipping the value of a single input bit.*

2.2.2 Restricted Network Congestion Games

In Chapter 5, we investigate the complexity of computing a Nash equilibrium in a restricted network congestion game. In a nutshell, restricted network congestion games are network congestion games where for each unweighted player there exits a set of edges which he is not allowed to use.

Definition 2.13. *An instance* $I = (\mathcal{G} = (V,E), \mathcal{N}, (E_i)_{i\in\mathcal{N}}, (a_i, b_i)_{i\in\mathcal{N}}, (d_e)_{e\in E})$ *of an n-player* restricted network congestion game *is characterized by five parameters:*

1. *a* directed or undirected network $\mathcal{G} = (V,E)$ *with a set of nodes* V *and a set of edges* E*,*
2. *a set of* n players $\mathcal{N} = \{1, \dots, n\}$*,*
3. *a set of* edges $E_i \subseteq E$ *for every player* $i \in \mathcal{N}$*,*
4. *a* source-sink pair (a_i, b_i) *for every player* $i \in \mathcal{N}$*, and*
5. *a* non-decreasing delay function $d_e : \mathbb{N} \to \mathbb{N}_0$ *for every edge* $e \in E$*.*

The strategy space for each player $i \in \mathcal{N}$ *is restricted to all simple paths connecting* a_i *and* b_i *via edges in* E_i*. We denote the* state of the game *by* $\mathsf{s} = (s_1, \dots, s_n)$*, where* s_i *for all* $i \in \mathcal{N}$*, denotes the strategy chosen by player* i *in* s*. Given some state* s *of the game, denote by* $n_e(\mathsf{s}) := |\{i \in \mathcal{N} \mid e \in s_i\}|$ *the number of players using edge* $e \in E$ *in* s*, which we denote as the* congestion on edge e *in* s*. Player* $i \in \mathcal{N}$*, given some state* s *of the game, incurs a* private cost *of*

$$\delta_i(\mathsf{s}) := \sum\nolimits_{e\in s_i} d_e(n_e(\mathsf{s}));$$

we use the term delay *synonymously. Players act selfishly and try to minimize their private cost. A state* s *of the game is a* Nash equilibrium *if no player can unilaterally deviate and strictly decrease his private cost.*

Rosenthal's potential function, which is polynomial-time computable, guarantees the existence of a Nash equilibrium in every restricted network congestion game [92]. This allows to formulate the problem of computing a Nash equilibrium in a given k-player restricted network congestion game on a directed (resp. undirected) network as a minimization $\mathcal{PLS}$ problem.

Definition 2.14 ((n)-RDNCG/(n)-RUNCG [4]). *An instance $I \in D_{(n)\text{-RDNCG}}$ (resp. $I \in D_{(n)\text{-RUNCG}}$) is an n-player restricted network congestion game*

$$(\mathcal{G} = (V,E), \mathcal{N}, (E_i)_{i\in\mathcal{N}}, (a_i,b_i)_{i\in\mathcal{N}}, (d_e)_{e\in E})$$

on a directed (resp. undirected) network $\mathcal{G} = (V,E)$ with $|\mathcal{N}| = n$ players. The set of feasible solutions $F_{(n)\text{-RDNCG}}(I)$ (resp. $F_{(n)\text{-RUNCG}}(I)$) consists of all states of the game. The cost of a solution $\mathsf{s} \in F_{(n)\text{-RDNCG}}(I)$ (resp. $\mathsf{s} \in F_{(n)\text{-RUNCG}}(I)$) is defined as the value of Rosenthal's potential function [92] at s, i. e.

$$c_{(n)\text{-RDNCG}}(\mathsf{s}, I) = c_{(n)\text{-RUNCG}}(\mathsf{s}, I) := \sum\nolimits_{e\in E} \sum\nolimits_{i=1}^{n_e(\mathsf{s})} d_e(i).$$

The neighborhood $N_{(n)\text{-RDNCG}}(\mathsf{s}, I)$ (resp. $N_{(n)\text{-RUNCG}}(\mathsf{s}, I)$) of a solution $\mathsf{s} \in F_{(n)\text{-RDNCG}}(I)$ (resp. $\mathsf{s} \in F_{(n)\text{-RUNCG}}(I)$) consists of all solutions $\mathsf{s}' \in F_{(n)\text{-RDNCG}}(I)$ (resp. $\mathsf{s}' \in F_{(n)\text{-RUNCG}}(I)$), which result from s by a player unilaterally deviating.

2.2.3 Local Search Versions of Generalized Maximum Satisfiability Problems

In Chapter 6, we investigate the complexity of computing locally optimal solutions of local search versions of the well-known Generalized Maximum Satisfiability problem (confer problem [LO6] in the book of Garey and Johnson [40] for a formal description). Essentially, the problems we study there are extensions of (h)-CNFSat to more generalized clauses over variables with larger valence; additionally, the appearance of each variable is a priori limited. For all local search versions of Generalized Maximum Satisfiability problems, we use the neighborhood structure where two solutions are mutual neighbors, if they can be transformed into each other by changing the assignment of a single variable. Except for Minimum Constraint Assignment, all problems we consider in this subsection are maximization problems.

Definition 2.15 ((p,q,r)-MCA [31]). *An instance $I \in D_{(p,q,r)\text{-MCA}}$ of (p,q,r)-Maximum Constraint Assignment (in short (p,q,r)-MCA), with $p,q,r \in \mathbb{N}$, is a set of constraints $\mathcal{C} = \{C_1, \ldots, C_m\}$ over a set of variables $\mathcal{X} = \{x_1, \ldots, x_n\}$. Each constraint $C_i(x_{i_1}, \ldots, x_{i_{p_i}}) \in \mathcal{C}$, with $i \in [m]$ and $p_i \leq p$, is a function $C_i : [r]^{p_i} \to \mathbb{N}_0$; p_i is also called* length *of C_i. Every variable $x \in \mathcal{X}$ appears in at most q constraints and takes values from $[r]$. The set of feasible solutions $F_{(p,q,r)\text{-MCA}}(I)$ consists of all assignments $\mathsf{a} : \mathcal{X} \to [r]$. The cost of a solution $\mathsf{a} \in F_{(p,q,r)\text{-MCA}}(I)$ is*

$$c_{(p,q,r)\text{-MCA}}(\mathsf{a}, I) := \sum\nolimits_{C_i \in \mathcal{C}} C_i(\mathsf{a}(x_{i_1}), \ldots, \mathsf{a}(x_{i_{p_i}})).$$

We denote by (p,q,r)-Minimum Constraint Assignment (in short (p,q,r)-MinCA) the minimization version of (p,q,r)-MCA. As a modification of (p,q,r)-MCA, we introduce (p,q,r)-VCA where p now denotes the maximum sum of the valences of the variables minus the number of variables in each constraint. Note that $(p,q,2)$-MCA equals $(p,q,2)$-VCA, by definition.

Definition 2.16 ((p,q,r)-VCA [31])**.** *An instance of* (p,q,r)-VALUE CONSTRAINT ASSIGNMENT *(in short* (p,q,r)-VCA*) is also a set of constraints* $\mathcal{C}$ *over a set of variables* $\mathcal{X}$*. Parameter* q *has the same meaning as in Definition 2.15. Parameter* r *denotes the maximum valence of the variables in* $\mathcal{X}$*. For each constraint* $C_i \in \mathcal{C}$*,* $\sum_{x \in C_i}(\mathsf{r}(x) - 1) \leq p$*, where* $\mathsf{r}(x)$ *denotes the valence of variable* $x \in \mathcal{X}$*.*

Definition 2.17 (k-partite Constraints [31])**.** *A set of constraints is* k-partite *if there exists a partition* $\mathsf{p} : \mathcal{X} \to [k]$ *of the variables such that for every two distinct variables* x, y *occurring in the same constraint,* $\mathsf{p}(x) \neq \mathsf{p}(y)$*. Denote by* (p,q,r)-MCA$_{k\text{-}par}$ *the subclass of all* (p,q,r)-MCA *instances where the set of constraints is k-partite.*

Definition 2.18 (Weighted Predicates [31])**.** *A* weighted predicate $P(x_1, \ldots, x_n)$ *over a set of r-valued variables* $\{x_1, \ldots, x_n\}$ *is a function* $P : [r]^n \to \{0,1\}$ *along with a* weight $\mathsf{w} \in \mathbb{N}_0$*. A weighted predicate can be viewed as a constraint with binary function values 0 and* w*.*

Let $x \in \{x_1, \ldots, x_n\}$ be a variable in predicate P. We say that setting variable x to a new value *improves* predicate P, if P was previously unsatisfied and is now satisfied. Setting variable x to a new value *violates* predicate P, if P was previously satisfied and is now unsatisfied. Note that every constraint can be decomposed into a set of weighted predicates sharing the same set of variables; the function value for a given assignment is the sum of the weighted predicates which are satisfied. In the following definition, we assume that each constraint is viewed as a set of predicates

Definition 2.19 (Hierarchical Set of Predicates [31])**.** *A set of predicates is* hierarchical *if the weight of each single predicate dominates the sum of the weights of all predicates of smaller weight. A set of constraints is hierarchical if the union of all constraints is hierarchical.*

2.2.4 Local Search Versions of Weighted Standard Set Problems

In Chapter 7, we investigate the complexity of computing locally optimal solutions of weighted standard set problems. All weighted standard set problems we present are local search versions of their respective decision problems, as included in the book of Garey and Johnson [40]. For an instance of a problem considered in this subsection, let $\mathcal{B}$ denote some finite set and let $\mathcal{C} = \{C_1, \ldots, C_n\}$ denote a set of subsets of $\mathcal{B}$. We refer to a set of sets also as a *collection*. Let $\mathsf{w}_\mathcal{C} : \mathcal{C} \to \mathbb{N}_0$ and $\mathsf{w}_\mathcal{B} : \mathcal{B} \times \mathcal{B} \to \mathbb{N}_0$. Denote by $m_\mathcal{B}$ and $m_\mathcal{C}$ positive integers with $m_\mathcal{B} \leq |\mathcal{B}|$ and $m_\mathcal{C} \leq |\mathcal{C}|$.

Unless otherwise mentioned, we use the *k-differ neighborhood*. Here, two solutions are mutual neighbors if they can be transformed into each other by adding, deleting, or exchanging at most k elements which describe a solution; more details are given in the respective definitions. Except for SETCOVER, all problems we consider in this subsection are maximization problems.

Definition 2.20 (W3DM-(p,q) [30])**.** *An instance* $I \in D_{\text{W3DM-}(p,q)}$ *of* WEIGHTED-3-DIMENSIONALMATCHING-(p,q) *(in short* W3DM-(p,q)*) is a pair* (n, w) *with* $n \in \mathbb{N}$

and w *is a function* $\mathsf{w} : [n]^3 \to \mathbb{N}_0$. *The components of triples are identified with boys, girls, and homes. The set of feasible solutions* $F_{\text{W3DM-}(p,q)}(I)$ *are all matchings of boys, girls, and homes, i. e. all* $\mathcal{S} \subseteq [n]^3$, *with* $|\mathcal{S}| = n$, $P_k(T_i) \neq P_k(T_j)$, *for all* $T_i, T_j \in \mathcal{S}$, $i \neq j$, *and* $k \in [3]$. *Given a solution* $\mathcal{S} \in F_{\text{W3DM-}(p,q)}(I)$, *the cost is*

$$c_{\text{W3DM-}(p,q)}(\mathcal{S}, I) := \sum\nolimits_{T \in \mathcal{S}} \mathsf{w}(T).$$

The neighborhood $N_{\text{W3DM-}(p,q)}(\mathcal{S}, I)$ *of a solution* $\mathcal{S} \in F_{\text{W3DM-}(p,q)}(I)$ *consists of all solutions* $\mathcal{S}' \in F_{\text{W3DM-}(p,q)}(I)$ *which result from* $\mathcal{S}$ *by replacing at most* p *triples and moving up to* q *boys or girls to new homes.*

Definition 2.21 (X3C-(k) [29]). *An instance* $I \in D_{\text{X3C-}(k)}$ *of* Exact-Cover-By-3-Sets-(k) *(in short* X3C-(k)*) is a tuple* $(\mathcal{C}, \mathsf{w}_{\mathcal{C}})$, *where* $\mathcal{C} = \{C_1, \ldots, C_n\}$ *is a collection of all 3-element sets of a finite set* $\mathcal{B}$, *with* $|\mathcal{B}| = 3q$ *for some* $q \in \mathbb{N}$. *The set of feasible solutions* $F_{\text{X3C-}(k)}(I)$ *are all collections* $\mathcal{S} \subseteq \mathcal{C}$ *such that every* $b \in \mathcal{B}$ *is in exactly one* $C_i \in \mathcal{S}$. *Given a solution* $\mathcal{S} \in F_{\text{X3C-}(k)}(I)$, *the cost is*

$$c_{\text{X3C-}(k)}(\mathcal{S}, I) := \sum\nolimits_{C_i \in \mathcal{S}} \mathsf{w}_{\mathcal{C}}(C_i).$$

The neighborhood $N_{\text{X3C-}(k)}(\mathcal{S}, I)$ *of a solution* $\mathcal{S} \in F_{\text{X3C-}(k)}(I)$ *consists of all solutions* $\mathcal{S}' \in F_{\text{X3C-}(k)}(I)$ *which differ from* $\mathcal{S}$ *in at most* k *sets.*

Definition 2.22 (SP-(k) [29]). *An instance* $I \in D_{\text{SP-}(k)}$ *of* SetPacking-(k) *(in short* SP-(k)*) is a triple* $(\mathcal{C}, \mathsf{w}_{\mathcal{C}}, m_{\mathcal{C}})$. *The set of feasible solutions* $F_{\text{SP-}(k)}(I)$ *are all collections* $\mathcal{S} \subseteq \mathcal{C}$ *with* $|\mathcal{S}| \leq m_{\mathcal{C}}$. *Given a solution* $\mathcal{S} \in F_{\text{SP-}(k)}(I)$, *the cost is*

$$c_{\text{SP-}(k)}(\mathcal{S}, I) := \sum\nolimits_{C_i \in \mathcal{S} \wedge \text{PwD}(C_i, \mathcal{S})} \mathsf{w}_{\mathcal{C}}(C_i),$$

where

$$\text{PwD}(C_i, \mathcal{S}) := \begin{cases} \mathtt{true} & \text{if } \forall C_j \in \mathcal{S} \setminus \{C_i\} : C_i \cap C_j = \emptyset \\ \mathtt{false} & \text{otherwise.} \end{cases}$$

The neighborhood $N_{\text{SP-}(k)}(\mathcal{S}, I)$ *of a solution* $\mathcal{S} \in F_{\text{SP-}(k)}(I)$ *consists of all solutions* $\mathcal{S}' \in F_{\text{SP-}(k)}(I)$ *which differ from* $\mathcal{S}$ *in at most* k *sets.*

Let us remark that the technical restriction on the maximum size of a feasible solution of an instance of SetPacking-(k) is crucial for our corresponding intractability result in Chapter 7.

Definition 2.23 (SSp-(k) [29]). *An instance* $I \in D_{\text{SSp-}(k)}$ *of* SetSplitting-(k) *(in short* SSp-(k)*) is a tuple* $(\mathcal{C}, \mathsf{w}_{\mathcal{C}})$. *Feasible solutions* $F_{\text{SSp-}(k)}(I)$ *are all partitions of* $\mathcal{B}$, *i. e. all pairs* (S_1, S_2) *with* $S_1, S_2 \subseteq \mathcal{B}$, $S_1 \cap S_2 = \emptyset$, *and* $S_1 \cup S_2 = \mathcal{B}$. *Given a solution* $\mathcal{S} = (S_1, S_2) \in F_{\text{SSp-}(k)}(I)$, *the cost is*

$$c_{\text{SSp-}(k)}(\mathcal{S}, I) := \sum\nolimits_{C_i \in \mathcal{C} \wedge \text{Split}(C_i, \mathcal{S})} \mathsf{w}_{\mathcal{C}}(C_i),$$

where

$$\textsc{Split}(C_i, \mathcal{S}) := \begin{cases} \texttt{true} & \text{if } (C_i \cap S_1 \neq \emptyset) \wedge (C_i \cap S_2 \neq \emptyset) \\ \texttt{false} & \text{otherwise.} \end{cases}$$

The neighborhood $N_{\text{SSP-}(k)}(\mathcal{S}, I)$ of a solution $\mathcal{S} \in F_{\text{SSP-}(k)}(I)$ consists of all solutions $\mathcal{S}' \in F_{\text{SSP-}(k)}(I)$, which result from $\mathcal{S}$ by at most k elements switching partition.

Definition 2.24 (SC-(k) [29])**.** *An instance $I \in D_{\text{SC-}(k)}$ of* SetCover-(k) *(in short* SC-(k)*) is a tuple $(\mathcal{C}, \mathsf{w}_{\mathcal{C}})$ such that $\bigcup_{C_i \in \mathcal{C}} C_i = \mathcal{B}$. The set of feasible solutions $F_{\text{SC-}(k)}(I)$ are all collections $\mathcal{S} \subseteq \mathcal{C}$ with $\bigcup_{C_i \in \mathcal{S}} C_i = \mathcal{B}$. Given a solution $\mathcal{S} \in F_{\text{SC-}(k)}(I)$, the cost is*

$$c_{\text{SC-}(k)}(\mathcal{S}, I) := \sum\nolimits_{C_i \in \mathcal{S}} \mathsf{w}_{\mathcal{C}}(C_i).$$

The neighborhood $N_{\text{SC-}(k)}(\mathcal{S}, I)$ of a solution $\mathcal{S} \in F_{\text{SC-}(k)}(I)$ consists of all solutions $\mathcal{S}' \in F_{\text{SC-}(k)}(I)$ which differ from $\mathcal{S}$ in at most k sets.

Definition 2.25 (TS-(k) [29])**.** *An instance $I \in D_{\text{TS-}(k)}$ of* TestSet-(k) *(in short* TS-(k)*) is a triple $(\mathcal{C}, \mathsf{w}_{\mathcal{B}}, m_{\mathcal{B}})$. Feasible solutions $F_{\text{TS-}(k)}(I)$ are all collections $\mathcal{S} \subseteq \mathcal{C}$ with $|\mathcal{S}| \in [m_{\mathcal{B}}]$. Given a solution $\mathcal{S} \in F_{\text{TS-}(k)}(I)$, the cost is*

$$c_{\text{TS-}(k)}(\mathcal{S}, I) := \sum\nolimits_{a,c \in \mathcal{B}, a \neq c \wedge \textsc{Test}(a,c,\mathcal{S})} \mathsf{w}_{\mathcal{B}}(a, c),$$

where

$$\textsc{Test}(a, c, \mathcal{S}) := \begin{cases} \texttt{true} & \text{if } \exists A, C \in \mathcal{S} : (a \in A \wedge c \notin A) \wedge (a \notin C \wedge c \in C) \\ \texttt{false} & \text{otherwise.} \end{cases}$$

The neighborhood $N_{\text{TS-}(k)}(\mathcal{S}, I)$ of a solution $\mathcal{S} \in F_{\text{TS-}(k)}(I)$ consists of all solutions $\mathcal{S}' \in F_{\text{TS-}(k)}(I)$ which differ from $\mathcal{S}$ in at most k sets.

Definition 2.26 (SB-(k) [29])**.** *An instance $I \in D_{\text{SB-}(k)}$ of* SetBasis-(k) *(in short* SB-(k)*) is a triple $(\mathcal{C}, \mathsf{w}_{\mathcal{C}}, m_{\mathcal{C}})$. The set of feasible solutions $F_{\text{SB-}(k)}(I)$ are all collections $\mathcal{S} = \{S_1, \ldots, S_{m_{\mathcal{C}}}\}$, where $S_i \in 2^{\mathcal{B}}$ with $|S_i| \in [h]$ for all $i \in [m_{\mathcal{C}}]$ and some fixed $h \in \mathbb{N}$ which is not part of the input. Given a solution $\mathcal{S} \in F_{\text{SB-}(k)}(I)$, the cost is*

$$c_{\text{SB-}(k)}(\mathcal{S}, I) := \sum\nolimits_{C_i \in \mathcal{C} \wedge \textsc{Set}(C_i, \mathcal{S})} \mathsf{w}_{\mathcal{C}}(C_i),$$

where

$$\textsc{Set}(C_i, \mathcal{S}) := \begin{cases} \texttt{true} & \text{if } \exists T \subseteq \mathcal{S} : C_i = \bigcup_{T_i \in T} T_i \\ \texttt{false} & \text{otherwise.} \end{cases}$$

The neighborhood $N_{\text{SB-}(k)}(\mathcal{S}, I)$ of a solution $\mathcal{S} \in F_{\text{SB-}(k)}(I)$ consists of all solutions $\mathcal{S}' \in F_{\text{SB-}(k)}(I)$ which differ from $\mathcal{S}$ in at most k sets.

Let us remark that with the technical restriction on the maximum size of each set in a feasible solution S of an instance of SetBasis-(k), the size of the neighborhood is again limited by a constant which is independent of the size of the input; the technical restriction that $\mathsf{S} \neq \emptyset$ is crucial for our corresponding intractability result in Chapter 7.

Definition 2.27 (HS-(k) [29]). *An instance $I \in D_{\text{HS-}(k)}$ of* HittingSet-(k) *(in short* HS-(k)*) is a triple $(\mathcal{C}, \mathsf{w}_{\mathcal{C}}, m_{\mathcal{B}})$. The set of feasible solutions $F_{\text{HS-}(k)}(I)$ are all sets $\mathsf{S} \subseteq \mathcal{B}$ with $|\mathsf{S}| \leq m_{\mathcal{B}}$. Given a solution $\mathsf{S} \in F_{\text{HS-}(k)}(I)$, the cost is*

$$c_{\text{HS-}(k)}(\mathsf{S}, I) := \sum\nolimits_{C_i \in \mathcal{C} \wedge \mathsf{S} \cap C_i \neq \emptyset} \mathsf{w}_{\mathcal{C}}(C_i).$$

The neighborhood $N_{\text{HS-}(k)}(\mathsf{S}, I)$ of a solution $\mathsf{S} \in F_{\text{HS-}(k)}(I)$ consists of all solutions $\mathsf{S}' \in F_{\text{HS-}(k)}(I)$ which differ from S in at most k elements.

Definition 2.28 (IP-(k) [29]). *An instance $I \in D_{\text{IP-}(k)}$ of* IntersectionPattern-(k) *(in short* IP-(k)*) is a triple $(A, B, \mathcal{D})$ consisting of two symmetric $n \times n$ matrices $A = (a_{ij})_{i,j \in [n]}$ and $B = (b_{ij})_{i,j \in [n]}$ with positive integer entries and a collection $\mathcal{D} = \{D_1, \ldots, D_\ell\}$ with $\ell \geq n$ over a finite set $\mathcal{B}$. The set of feasible solutions $F_{\text{IP-}(k)}(I)$ are all vectors $\mathcal{E} = (E_1, \ldots, E_n)$ with $E_i \in \mathcal{D}$ for all $i \in [n]$. Given a solution $\mathsf{E} \in F_{\text{IP-}(k)}(I)$, the cost is*

$$c_{\text{IP-}(k)}(\mathsf{E}, I) := \sum\nolimits_{i,j \in [n], i \leq j \wedge |E_i \cap E_j| = a_{ij}} b_{ij}.$$

The neighborhood $N_{\text{IP-}(k)}(\mathsf{E}, I)$ of a solution $\mathsf{E} \in F_{\text{IP-}(k)}(I)$ consists of all solutions $\mathsf{E}' \in F_{\text{IP-}(k)}(I)$ which differ from E in at most k vector entries.

Definition 2.29 (CC-(k) [29]). *An instance $I \in D_{\text{CC-}(k)}$ of* ComparativeContainment-(k) *(in short* CC-(k)*) is a triple $(\mathcal{C}, \mathcal{D}, \mathsf{w})$ consisting of two collections $\mathcal{C} = \{C_1, \ldots, C_n\}$, and $\mathcal{D} = \{D_1, \ldots, D_\ell\}$ over a finite set $\mathcal{B}$, and a function $\mathsf{w} : \mathcal{C} \cup \mathcal{D} \to \mathbb{N}_0$. The set of feasible solutions $F_{\text{CC-}(k)}(I)$ are all sets $\mathsf{S} \subseteq \mathcal{B}$. Given a solution $\mathsf{S} \in F_{\text{CC-}(k)}(I)$, the cost is*

$$c_{\text{CC-}(k)}(\mathsf{S}, I) := \mathsf{W} + \sum\nolimits_{C_i \in \mathcal{C} : \mathsf{S} \subseteq C_i} \mathsf{w}(C_i) - \sum\nolimits_{D_i \in \mathcal{D} : \mathsf{S} \subseteq D_i} \mathsf{w}(D_i),$$

where $\mathsf{W} \geq \sum_{D_i \in \mathcal{D}} \mathsf{w}(D_i)$, in order for $c_{\text{CC-}(k)}(\mathsf{S}, I) \geq 0$ for each $\mathsf{S} \in F_{\text{CC-}(k)}(I)$. The neighborhood $N_{\text{CC-}(k)}(\mathsf{S}, I)$ of a solution $\mathsf{S} \in F_{\text{CC-}(k)}(I)$ consists of all solutions $\mathsf{S}' \in F_{\text{CC-}(k)}(I)$ which differ from S in at most k elements.

Chapter 3

Related Work

In this chapter, we survey the research on the complexity of computing locally optimal solutions in predominantly chronological order. We first focus on *early results* in Section 3.1, starting with the fundamental problem CIRCUIT/FLIP. We then continue by presenting results for the TRAVELING SALESMAN PROBLEM, MAXCUT, and local search versions of SATISFIABILITY problems, weighted set problems, and graph problems. Most of the pioneering results are from Johnson, Papadimitriou, and Yannakakis [56, 87] or from Schäffer and Yannakakis [95, 109]. We close by surveying recent results from *game theory* in Section 3.2, considering the complexity of computing pure Nash equilibria. Here, we concentrate on the research on *congestion games*, which sparked the *interconnection* between *local search* and *game theory.*

For further known $\mathcal{PLS}$-complete problems, we refer the reader to the works of Alekseeva et al. [7], Brandt et al. [17], Klauck [63], Kochetov and Ivanenko [66], Papadimitriou et al. [87], Prokopyev et al. [90], Vredeveld and Lenstra [107]. An excellent *overview* of local search algorithms in general, their applications and known tractability results, is presented in the books of Aarts et al. [2] and Aarts and Lenstra [1], of which the first one contains a *list of $\mathcal{PLS}$-complete problems* known so far. A survey of the *quality of solutions* obtained via local search is given by Angel [8]. Unless otherwise mentioned, we assume that all numbers are integers, for the remainder of this chapter.

3.1 Early Results

The early results in the field of $\mathcal{PLS}$ were motivated by determining the complexity of computing locally optimal solutions of well-known hard combinatorial optimization problems. Here, the neighborhood structure is superimposed by successful local search algorithms, such as the ones presented in Section 1.2.2.

3.1.1 Circuit/Flip

The first $\mathcal{PLS}$ problem which was proven to be $\mathcal{PLS}$-complete is CIRCUIT/FLIP. In a nutshell, the intractability proof given by Johnson, Papadimtriou, and Yannakakis [56] involves three intermediate $\mathcal{PLS}$ reductions:

1. First, the authors reduce an instance I of an arbitrary problem $L \in \mathcal{PLS}$ to an instance I' of an intermediate problem $L' \in \mathcal{PLS}$; L' only differs from L in

the neighborhood structure, as in L' no feasible solution has more than one neighbor. The design of the reduction is such that each solution $\mathsf{s}' \in F_{L'}(I')$ has at most one neighbor and local optima coincide. For this, the single neighbor of each solution $\mathsf{s} \in F_{L'}(I')$ is defined to be the output of $\textsc{Improve}_L(\mathsf{s}, I)$.

2. Without loss of generality, assume that all solutions $\mathsf{s} \in F_L(I)$ are binary strings of length at most a polynomial in the length of I; denote the latter one by $p(|I|)$. Second, the authors reduce L' to another problem $L'' \in \mathcal{PLS}$ which has the set of solutions of L', but every bit string of length $p(|I|)$ is now a feasible solution; two solutions are mutual neighbors if they differ in a single bit. Note that this exactly matches the neighborhood structure of Circuit/Flip. The cost function is defined such that every minimum length sequence of bit-flips between two solutions $\mathsf{s}, \mathsf{t} \in F_{L'}(I')$ yields an improving path $\mathsf{s} \rightsquigarrow \mathsf{t}$ if and only if $\mathsf{t} = N_{L'}(\mathsf{s}, I')$. The crucial idea here is to assign every intermediate solution u on each such minimum length sequence $\mathsf{s} \rightsquigarrow \mathsf{t}$ the cost of t scaled by some factor plus the *Hamming distance* between u and t; call the resulting instance I''.

3. The final reduction from L'' to Circuit/Flip simply involves constructing a feedback-free boolean circuit which computes the cost function of I''.

For all technical details, we refer the reader to the original paper [56].

Notably, this reduction is tight [87]. Hence, the exclusiveness of exponentially long sequences of improving steps in $\mathcal{PLS}$ problem Exponential is preserved by the reduction; subsequently, Circuit/Flip possesses the all-exp property. Furthermore, the tightness of the reduction implies that the standard algorithm problem for Circuit/Flip is $\mathcal{PSPACE}$-complete, since the $\mathcal{PLS}$ problem whose standard algorithm problem is $\mathcal{PSPACE}$-complete can be directly simulated using Circuit/Flip. By Definition 2.9 in Chapter 2, Circuit/Flip is tight $\mathcal{PLS}$-complete. Hence, problems shown to be tight $\mathcal{PLS}$-complete via a sequence of tight reductions from Circuit/Flip are again tight $\mathcal{PLS}$-complete, implying that these problems possess the all-exp property and their standard algorithm problem is $\mathcal{PSPACE}$-complete.

3.1.2 Traveling Salesman Problem

As outlined in Section 1.2.2, local search algorithms have been applied with huge success to compute approximate solutions of the Traveling Salesman Problem on random Euclidean instances [55]. On the negative side, finding a locally optimal solution of the Traveling Salesman Problem, where the neighborhood structure is superimposed by the well-known k-Opt heuristic [71] for some $k \gg 1,000$ is $\mathcal{PLS}$-complete [67]. In the k-Opt neighborhood, two solutions are mutual neighbors if they differ in at most k edges. Computing a locally optimal solution with respect to the Lin-Kernighan heuristic [72] (confer Johnson and McGeoch [55] for a formal definition) is also $\mathcal{PLS}$-complete [85, 87].

3.1.3 MaxCut

The MaxCut problem is known to be tight $\mathcal{PLS}$-complete [95]. Let us remark that in the reduction given by Schäffer and Yannakakis [95], the maximum degree of the constructed graph is required to be unbounded. Recently, it was shown that a maximum degree of five is sufficient such that MaxCut is $\mathcal{PLS}$-complete [104]. If the maximum degree of the input graph is at most three, then every sequence of improving steps has at most a length which is quadratic in the size of the input [89]. Hence, locally optimal solutions can be computed in polynomial time via successive improvements. This does no longer hold for graphs of maximum degree four, as it was recently shown that in this case MaxCut has the all-exp property [77].

3.1.4 Local Search Versions of Satisfiability Problems

In their seminal paper, Johnson, Papadimtriou, and Yannakakis [56] conjecture that for a problem to be $\mathcal{PLS}$-complete, the problem of verifying local optimality is required to be $\mathcal{P}$-complete. Krentel [68] disproves this conjecture by showing that CNF-Satisfiability is $\mathcal{PLS}$-complete, though the problem of verifying local optimality can be solved in $\mathcal{LOGSPACE}$. Let us remark that in the reduction given by Krentel [68], the length of each clause is required to be unbounded. In a succeeding paper, Krentel [67] outlines that $(4, 3, 3)$-MCA is $\mathcal{PLS}$-complete. This result can be extended to (h)-CNFSat for some fixed maximum length $h \in \mathbb{N}$ of each clause where simultaneously the maximum appearance of each variable in every instance is fixed to some $q \in \mathbb{N}$ which is independent of the instance [67, 104]. Furthermore, Positive-NotAllEqual-2-Flip, a reformulation of MaxCut, is known to be $\mathcal{PLS}$-complete, along with Maximum 2-Satisfiability [95]. For any generalized local search version of Satisfiability, the dichotomy theorem by Chapdelaine and Creignou [20] states that the corresponding $\mathcal{PLS}$ problem is either in $\mathcal{P}$ or $\mathcal{PLS}$-complete; this is in spirit with the dichotomy theorem presented by Schaefer [94].

3.1.5 Weighted Set Problems and Graph Problems

The knowledge of the complexity of computing locally optimal solutions of weighted set problems is rather limited. Known results are based on results for local search versions of *graph problems* that can be reformulated as set problems. Besides the *edge-weighted* graph problems in $\mathcal{PLS}$ outlined above and considered in the works of Johnson et al. [56], Papadimitriou et al. [87], Schäffer and Yannakakis [95], Vredeveld and Lenstra [107], we now present the following two representatives of *vertex-weighted* graph problems in $\mathcal{PLS}$.

The Independent Dominating h-Set problem is a minimization local search version of Dominating Set (confer [GT2] in the book of Garey and Johnson [40] for a formal definition). In the Independent Dominating h-Set problem, the input consists of a vertex-weighted undirected graph of maximum degree h. Feasible solutions are all binary assignments for the vertices, such that no two adjacent vertices may be assigned 1 and for each vertex which is assigned 0, there exists an adjacent vertex which is assigned 1. The cost of an assignment is the sum of the weights of the vertices which are assigned 1. Given an assignment, its neighborhood consists of all assignments, where at most k vertices switch their assignment. The Independent Dominating h-Set problem is known to be $\mathcal{PLS}$-complete for sufficiently large, but constant h and k [63].

Shimozono [97] provides a more generalized intractability result for vertex-weighted graph problems by showing $\mathcal{PLS}$-completeness of the maximization problem Weighted Greedy Maximal-Π. In the Weighted Greedy Maximal-Π problem, the input consists of a vertex-weighted graph. The set of feasible solutions contains all subsets of the vertices that induce a subgraph which satisfies a fixed graph property Π; property Π is required to be checkable in polynomial time. The cost of a solution is the sum of the weights of its vertices; the neighborhood structure is superimposed by an algorithm that finds a greedy maximal subgraph. If Π is any nontrivial and hereditary graph property, then Weighted Greedy Maximal-Π is known to be $\mathcal{PLS}$-complete [97].

3.2 Recent Results and the Connection to Game Theory

Recently, the field of local search has attracted additional attention from game theory, considering the complexity of computing a pure Nash equilibrium. In this section, we only consider pure Nash equilibria, unless otherwise mentioned; thus, we omit pure for sake of readability. In the following, we survey results considering the complexity of computing a Nash equilibrium in a given congestion game, as introduced in Subsection 1.2.5. For $\mathcal{PLS}$-completeness results in other games, we refer the reader to the works of Ackermann et al. [5], Dunkel and Schulz [32], Goemans et al. [41]. For related work including other aspects of decentralized autonomous systems like congestion games such as the price of anarchy or network design problems, confer the book of Nisan et al. [81].

3.2.1 Computing Nash Equilibria in Unweighted Congestion Games

First, we consider unweighted congestion games. In their seminal paper, Fabrikant, Papadimitriou, and Talwar [34] settle the complexity of computing a Nash equilibrium. Finding a Nash equilibrium in a symmetric network congestion game is polynomial-time solvable by reduction to the Min-Cost-Flow problem [34]. Computing a Nash equilibrium in a symmetric congestion game and in an asymmetric network congestion game is tight $\mathcal{PLS}$-complete; hence, these games possess the all-exp property [34]. Notably, neither the numbers of players nor the number of resources per strategy is a priori bounded in both reductions. A recent result by Tscheuschner [104] implies that the above intractability results of Fabrikant et al. [34] can be extended to hold for at most five resources per strategy. Despite the polynomial-time computability of Nash equilibria in symmetric network congestion games, an asymmetric network congestion game having the all-exp property can be embedded; hence, the class of symmetric network congestion games possess the all-exp property [5]. The $\mathcal{PLS}$-completeness result for asymmetric network congestion games was refined to hold for undirected networks even if all latency functions are linear, including an elegant proof for the $\mathcal{PLS}$-completeness of directed asymmetric network congestion games [5]. If the strategy space of each player consists of the bases of a matroid over the set of resources, then Nash equilibria can be efficiently computed, using a best-response Nash dynamics. The matroid property is a sufficient and necessary condition on the combinatorial structure of the players' strategy spaces to guarantee fast convergence to Nash equilibria [5]. On the other hand, if the strategies of a congestion game fulfill a so called $(1,2)$-exchange property, then the problem of finding a Nash equilibrium has the is-exp property [5].

3.2.2 Approximate Nash Equilibria in Unweighted Congestion Games

Since in many cases, the computation of a Nash equilibrium is as hard as finding a local optimum for any problem in $\mathcal{PLS}$, one might hope that the computation of approximate Nash equilibria yields a significant performance improvement. A δ-approximate Nash equilibrium is a state of the game in which no player can unilaterally improve his delay by a factor of at least δ. The existence of a δ-approximate Nash equilibrium is guaranteed by Rosenthal's potential function and may be computed by adapting the Nash dynamics to now incorporate unilateral deviations which improve the delay of the deviating player by a factor of at least δ. The hope for an efficient computation of δ-approximate Nash equilibria might be additionally fueled by the existence of an FPTAS for computing approximate local optima for all linear combinatorial optimization problem in $\mathcal{PLS}$ [82]. In particular, the authors provide a rule for choosing improving steps that lead to an approximate solution within a polynomial number of steps. Unfortunately this notion of approximation is not sufficient for approximate Nash equilibria, since each selfish player is ignorant of the potential function. There might be solutions which are up to a factor of δ close to a local optimum, considering the potential function, but a single player might still have an incentive of δ or more to unilaterally deviate.

As outlined in Subsection 1.2.5, if all delay functions in unweighted congestion games are polynomials, then Nash equilibria can be efficiently computed. The question arises to which extent the delay functions can be generalized such that polynomial-time convergence to approximate Nash equilibria is still guaranteed. A congestion game satisfies the α-bounded jump condition, if for every resource with at least one player, the addition of a single player increases the delay by at most a factor of α. Note that delay functions satisfying the α-bounded jump condition are more general than polynomial delay functions, but not exponential. This condition does not trim the inherent complexity of congestion games, as finding a Nash equilibrium in a symmetric congestion game satisfying the α-bounded jump condition with $\alpha = 2$ is $\mathcal{PLS}$-complete [22]. On the other hand, computing a δ-approximate Nash equilibrium in a symmetric congestion game, where each edge satisfies the α-bounded jump condition, is polynomial-time computable, since the sequence of slightly restricted

δ-selfish steps converges within a number of steps that is polynomial in the number of players, α, and δ^{-1}. The problem of computing a Nash equilibrium in an asymmetric congestion game satisfying the α-bounded jump condition via δ-best improvement steps possesses the all-exp property [98]. Hence, the positive result is restricted to symmetric congestion games. Computing a δ-approximate Nash equilibrium in an arbitrary congestion game is $\mathcal{PLS}$-complete, for every polynomial-time computable approximation factor δ [98]. As the reduction by Skopalik and Vöcking [98] is tight, the problem of computing a δ-approximate Nash equilibrium in an arbitrary congestion game has the all-exp property.

3.2.3 Player-Specific (Singleton) Congestion Games

Now, we turn our attention to a variation of congestion games known as congestion games with *player-specific latency functions*, originally introduced by Milchtaich [75]. While in congestion games, all players share the same delay function for a resource, player-specific delay functions rather emphasize the players' personal preferences for certain resources. Note that this setting also allows to model that each player may only use a certain subset of the resources. In the model of *singleton congestion games*, every strategy of each player consists of a single resource. Symmetric singleton congestion games are then a subclass of network congestion games where players route their demand through a simple network of parallel edges between two nodes s and t.

First, we consider symmetric singleton congestion games with player-specific latency functions. In general, unweighted symmetric singleton congestion games with player-specific delay functions possess Nash equilibria, but may not necessarily have a potential function, even in the case of three players [75]. Moreover, this result is tight since in the case of two players, the Nash dynamics converges [75]. In the case of weighted players and player-specific non-decreasing latency functions, Nash equilibria might not exist for games with three players [75]. Unweighted symmetric singleton congestion games with player-specific linear delay functions without a constant term possess a potential function and therefore also Nash equilibria [39]. This result does not extend to player-specific linear delay functions. Concatenations of two unweighted symmetric singleton congestion games possess Nash equilibria [76] but may not necessarily have a potential function [39]. The special class of unweighted symmetric singleton congestion games with player-specific constants, where all player-specific delay functions are composed of a common resource-specific delay function, but each player may have a player-specific constant for the particular resource, possesses a potential function [74].

Now, we consider general (singleton) congestion games with player-specific latency functions. Computing a Nash equilibrium in a symmetric network congestion game with player-specific constants is $\mathcal{PLS}$-hard [74]. Restricted network congestion games can also be regarded as network congestion games with player-specific constants zero or infinity. For restricted network congestion games with three players, finding a Nash equilibrium is $\mathcal{PLS}$-complete [4]. On the other hand, computing Nash equilibria in singleton congestion games can be efficiently done if all delay function are the identity function [38]. Notably, all delay functions are required to be the identity function; for arbitrary linear delay functions, the complexity is unknown.

3.2.4 Symmetric Singleton Congestion Games

Here, we only consider symmetric singleton congestion games where for each resource, all players share the same latency function. In case all latency functions are non-decreasing, Nash equilibria are guaranteed to exist [36]. If all latency functions are linear, then Nash equilibria can be efficiently computed using Graham's LONGEST PROCESSING TIME (in short LPT) algorithm [36, 44]. Notably, these games also possess the is-exp property, even for the best-response Nash dynamics [35]. The simplicity of the network seems in most cases to allow for efficient algorithms which compute Nash equilibria, but the situation somewhat changes, once weighted players may form arbitrary non-fixed coalitions. If players may form such coalitions of size at most 8 and in each improving step of a coalition, the maximum cost of its members decreases, then computing a Nash equilibrium is $\mathcal{PLS}$-complete [27].

3.2.5 Related: Equilibrium Search and $\mathcal{PPAD}$

The previous subsections outline that the computation of a Nash equilibrium in a congestion game can be treated as a local search problem. However, there are finite games for which only the existence of a *mixed* Nash equilibrium is guaranteed by Nash's theorem [80]. In a mixed Nash equilibrium, each player chooses a probability distribution over his set of strategies such that a unilateral deviation to a different probability distribution does not strictly improve his expected private cost. Note that pure Nash equilibria that have been considered so far are mixed Nash equilibria where each player commits to a single strategy. BIMATRIX, the problem of computing a mixed Nash equilibrium of a 2-player game with rational utilities for the players, is a famous example of a finite game in which in general only the existence of mixed Nash equilibria is guaranteed.

Already in 1964, Lemke and J. T. Howsen [70] developed an algorithm that computes a mixed Nash equilibrium for 2-player games. Their algorithm works similar to the SIMPLEX algorithm and is comparably successful in practical applications. However, it is known that there are 2-player games, for which the LEMKE-HOWSON algorithm—even in the best case—takes an exponential number of steps before reaching a mixed Nash equilibrium [93]. Motivated by proofs that show the existence of solutions via the lemma that "every graph has an even number of odd degree nodes", Papadimitriou [86] introduced the complexity class $\mathcal{PPAD}$ (short for "Polynomial Parity Argument in a Directed graph"). The problems in this class are defined via implicitly given, exponentially large directed graphs consisting of directed paths, cycles, and single nodes, where one artificial source is given. The problems ask for an endpoint of a path, i. e. a source or a sink, distinct from the given source. Shapley [96] shows that the possible steps of the LEMKE-HOWSON algorithm induce a graph with the above properties and therefore Bimatrix $\in \mathcal{PPAD}$ [106].

Significant progress in the classification of the complexity of computing mixed Nash equilibria for games with a finite number of players was achieved by Daskalakis, Goldberg, and Papadimitriou [25], who show that computing a mixed Nash equilibrium for four-player games is $\mathcal{PPAD}$-hard. Using their construction, Chen, Deng, and Teng [21] show that Bimatrix is $\mathcal{PPAD}$-complete. However, as for congestion games, the approximation of mixed Nash equilibria of 2-player games appears no easier than the computation of a mixed Nash equilibrium itself, since the existence of an FPTAS would imply that $\mathcal{PPAD}$ is in $\mathcal{P}$ [21]. For an overview of further results related to $\mathcal{PPAD}$, we refer the reader to the survey by Yannakakis [110].

Chapter 4

Our Contribution

In this chapter, we *summarize the main results*, we present in this thesis. Here, Section 4.1 pools our main results on the tractability of computing Nash equilibria in restricted network congestion games, which we investigate in Chapter 5. Section 4.2 lists our results on the complexity of computing locally optimal solutions of the MAXIMUM CONSTRAINT ASSIGNMENT problem which we study in Chapter 6. Section 4.3 summarizes our results on the tractability of computing locally optimal solutions of weighted standard set problems which we investigate in Chapter 7.

Our contribution consists in large parts of intractability results, which we prove along a chain of $\mathcal{PLS}$ reductions. A *graphical overview* of the main reductions we present is given in Figure 4.1. Here, a directed arrow between A and B denotes a $\mathcal{PLS}$ reduction from A to B. The endpoints of the arrows either refer to $\mathcal{PLS}$ problems or to the entire class $\mathcal{PLS}$, as is the case for the upmost reduction, showing the $\mathcal{PLS}$-completeness of CIRCUIT/FLIP. Arrow labels either refer to the publication where the corresponding reduction is given or to the corresponding theorem or lemma where we present the proof of $\mathcal{PLS}$-completeness. Braces on the left hand side of the figure indicate the corresponding chapters, where the $\mathcal{PLS}$ reductions can be found. For sake of readability, the endpoints of braces have been extended across the figure.

4.1 Computing Nash Equilibria in Two-Player Restricted Network Congestion Games

In Chapter 5, we investigate the complexity of computing Nash equilibria in restricted network congestion games involving two players only, as introduced in Subsection 2.2.2. As our main result, we show the *intractability of the corresponding $\mathcal{PLS}$ problem* for *directed and undirected* networks.

Theorem 5.1. (2)-RDNCG *and* (2)-RUNCG *are tight $\mathcal{PLS}$-complete for graphs of degree at most three.*

Our results are *optimal* in the sense that for one player, computing a Nash equilibrium reduces to finding a shortest path, which is polynomial-time computable. To the best of our knowledge our result is the single $\mathcal{PLS}$-intractability result for games with two players only.

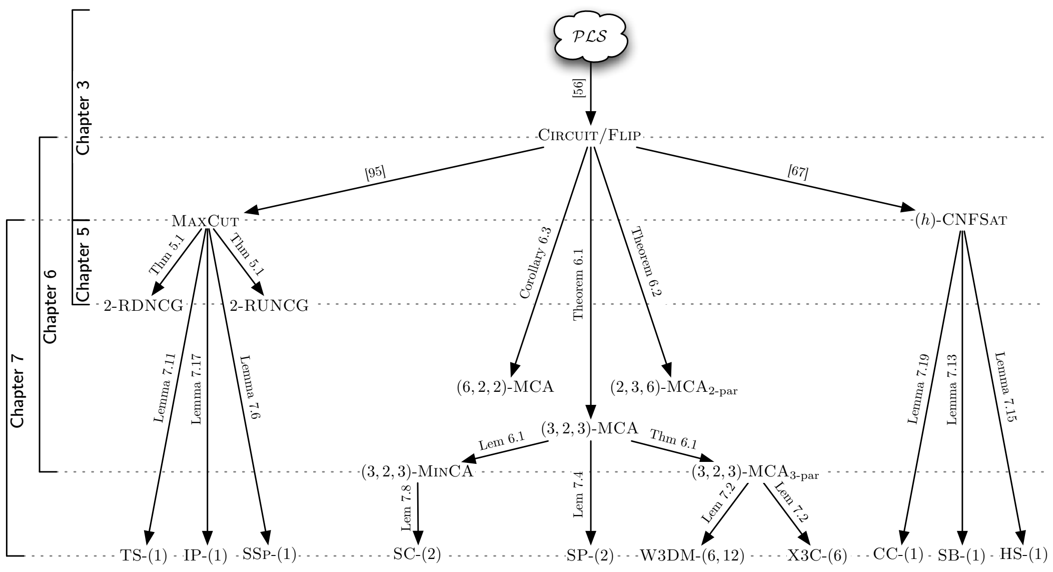


Figure 4.1: Graphical overview of the main $\mathcal{PLS}$ reductions, we present in this thesis.

4.2 On the $\mathcal{PLS}$-Complexity of Maximum Constraint Assignment

In Chapter 6, we study the complexity of computing locally optimal solutions of the Maximum Constraint Assignment problem. We first show that all intractability results for (p, q, r)-MCA$_{k\text{-par}}$ extend to (p, q, r)-MinCA$_{k\text{-par}}$ for all $p, q, r, k \in \mathbb{N}$.

Lemma 6.1. *(p, q, r)-MCA$_{k\text{-par}}$ $\leq_{pls}$ (p, q, r)-MinCA$_{k\text{-par}}$ for all $p, q, r, k \in \mathbb{N}$, using a tight reduction.*

We then focus on the hardness of $(3, 2, *)$-MCA$_{3\text{-par}}$ and $(2, 3, *)$-MCA$_{2\text{-par}}$, as introduced in Subsection 2.2.3 and *optimize for minimum valence* of the variables. As our main result, we show the *intractability* of $(3, 2, 3)$-MCA$_{3\text{-par}}$ and $(2, 3, 6)$-MCA$_{2\text{-par}}$, using *tight* reductions.

Theorem 6.1. Circuit/Flip $\leq_{pls}$ $(3, 2, 3)$-MCA$_{3\text{-par}}$ *using a tight reduction.*

Theorem 6.2. Circuit/Flip $\leq_{pls}$ $(2, 3, 6)$-MCA$_{2\text{-par}}$ *using tight reductions*

The above intractability results are *optimal* in the sense that $(2, 2, r)$-MCA is solvable in polynomial time for every $r \in \mathbb{N}$. Additionally, we present a *general technique* to simulate arbitrary (p, q, r)-VCA-instances with VCA-instances over solely *binary* variables. We present a reduction in Subsection 6.5, where the sum of the valences of the variables minus the number of variables in each constraint remains constant. Our construction eventually proves the following theorem:

Theorem 6.3. *For all $p, q \in \mathbb{N}$ and $r \geq 3$, (p, q, r)-VCA $\leq_{pls}$ $(p, q, 2)$-VCA using a tight reduction.*

As a direct consequence of Theorem 6.3, we obtain the tight $\mathcal{PLS}$-completeness of $(6,2,2)$-MCA.

Corollary 6.3. CIRCUIT/FLIP $\leq_{pls}$ $(6,2,2)$-MCA *using a tight reduction.*

For the proofs of Theorem 6.1 and Theorem 6.2, we use a technique introduced by M. W. Krentel [67]. Krentel outlines that $(4,3,3)$-MCA is $\mathcal{PLS}$-complete. His construction is not complete and we do not think that it can be extended to obtain a tight reduction. We *sharpen essential parts* of the reduction, *reduce* both parameters p and q and also obtain *tightness.* We think that reducing the parameters p and q is important. The tight $\mathcal{PLS}$-completeness of $(3,2,3)$-MCA$_{\text{3-par}}$ and especially the fact that each variable appears in at most two constraints will prove crucial in Chapter 7 in order to demarcate the tractability of SETPACKING and SETCOVER; furthermore, the tight $\mathcal{PLS}$-completeness of $(3,2,3)$-MCA$_{\text{3-par}}$ will also play a vital role in order to lower the bounds on the intractability of other local search versions of weighted standard set problems.

Moreover, many problems can be modeled in a natural way as SATISFIABILITY problems. In the $\mathcal{NP}$-world, this has been exploited in a large number of cases. We think that also for $\mathcal{PLS}$ problems it will help to *sharpen the boundary* between $\mathcal{PLS}$-complete and solvable in polynomial time.

4.3 On the Complexity of Local Search for Weighted Standard Set Problems

In Chapter 7, we investigate the tractability of computing locally optimal solutions for the weighted standard set problems introduced in Subsection 2.2.4. We show that for most of these problems, computing a locally optimal solution is *$\mathcal{PLS}$-complete* for the *1-differ neighborhood.* This means, that the respective problems are already intractable, when one element describing the solution is allowed to be added, deleted, or exchanged for another element which is not part of the solution. For SETPACKING-(k) and SETCOVER-(k), we delimit the tractability of computing locally optimal solutions for the 2-differ neighborhood. As our main result, we prove the following two theorems:

Theorem 7.1. *The problems* SSP-(k), TS-(k), HS-(k), SB-(k), IP-(k), *and* CC-(k) *are tight $\mathcal{PLS}$-complete for all $k \geq 1$. The problems* SP-(k) *and* SC-(k) *are tight $\mathcal{PLS}$-complete for all $k \geq 2$. The problems* W3DM-(k,ℓ) *and* X3C-(k) *are tight $\mathcal{PLS}$-complete for all $k \geq 6$ and $\ell \geq 12$.*

Theorem 7.2. *The problems* SP-(1) *and* SC-(1) *are polynomial-time solvable.*

A *graphical representation* of our results in Chapter 7 is given in Figure 4.2. Here, arrows represent problems. From bottom to top, they indicate problems which are polynomial-time computable; from top to bottom, they indicate $\mathcal{PLS}$-complete problems. Additionally, a dashed line denotes the border between polynomial-time computable and $\mathcal{PLS}$-complete problems. Note that this dashed line is not extended

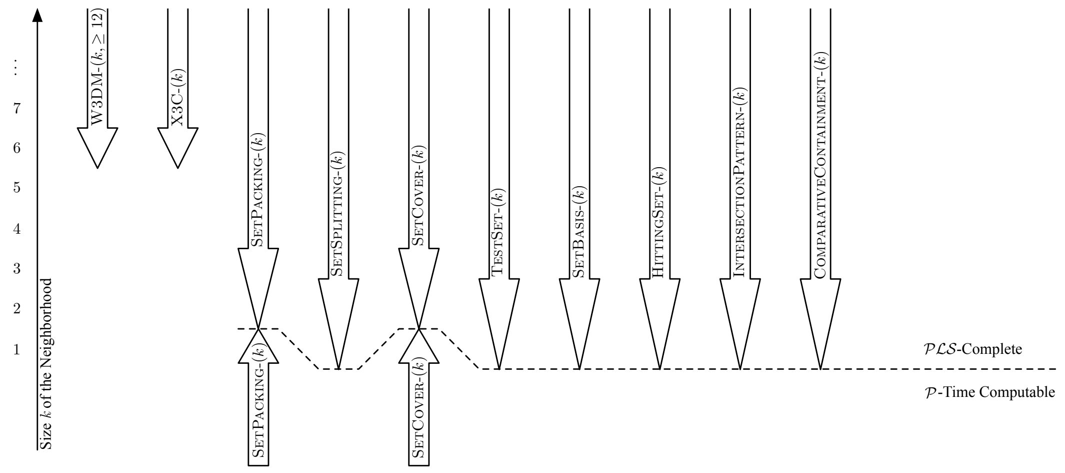


Figure 4.2: Graphical overview of our results on the complexity of local search for weighted standard set problems, presented in Chapter 7.

to the first two problems, since our results do not yield an exact border in these cases. Let us stress that all reductions we present are *tight* in the sense of Schäffer and Yannakakis [95].

Differing from previous known results, which we outlined in Subsection 3.1.5, our results for weighted standard set problems do not rely on a reformulation of local search versions of graph problems. To the best of our knowledge, the results we present are one of the first $\mathcal{PLS}$ results directly shown for these problems. Moreover, they are one of the very few $\mathcal{PLS}$ results for weighted standard set problems, as intensively studied in the literature.

4.4 A Note on the Presentation of Our Reductions

Most of the reductions we present in this thesis tend to be rather technically involved constructions.

Presentation of Techniques and Core Ideas For sake of presentation, we first present the *underlying techniques* used in Chapters 5–7 in separate subsections of the respective chapters. The core idea for the proof of intractability of (2)-RDNCG and (2)-RUNCG is given in Subsection 5.1.1. We describe the technique from Krentel [67], which we extend and refine for the intractability proofs of $(3, 2, 3)$-MCA$_{\text{3-par}}$ and $(2, 3, 6)$-MCA$_{\text{2-par}}$, in Subsection 6.2. The intractability proofs of locally optimal solutions for the weighted standard set problems we investigate, follow a technique which we present in Subsection 7.1. In Chapter 7, we additionally present the *core ideas* of each reduction *in a nutshell* first, before stepping into the corresponding construction.

The Structure of our Presentation Each reduction includes the presentation of the reduction function Φ and the solution mapping Ψ. The latter one often involves a subclass of feasible solutions for the problem under investigation which we define as *standard solutions* or solutions which are *consistent* for a certain property. In spirit with the widely accepted “quod erat demonstrandum” symbol ($\square$), we choose to *indicate the end of each reduction by a circle* ($\bigcirc$), for sake of readability. For rather involved proofs, we present a *roadmap* before stepping into the proof of correctness and tightness of the respective reduction. Essentially, the proofs of correctness can be split into two parts. In the first part, we show that every locally optimal solution is a standard solution or consistent for the respective property. In the second, part, we show that locally optimal solutions in the constructed instance correspond to locally optimal solutions of the input instance; here, the first part allows to solely focus on standard or consistent solutions. Generally and for sake of readability, we drop the index of the problem in identifiers in our presentations, where the problem is clear from the context.

Chapter 5

Computing Nash Equilibria in Two-Player Restricted Network Congestion Games

In this chapter, we show that computing a Nash equilibrium in a restricted network congestion game is already $\mathcal{PLS}$-complete for two players. The result holds for directed networks and for undirected networks. Our reduction from MaxCut only requires two gadgets and is conceptionally simpler than the reduction presented by Ackermann and Skopalik [4] which shows the intractability of computing Nash equilibria in restricted network congestion games with three players. The results we present are optimal, since for one player, computing a Nash equilibrium reduces to finding a shortest path, which is polynomial-time computable. Let us remark that the complexity of computing a Nash equilibrium in a standard *network* congestion game with a fixed number of players remains unsettled.

This chapter is organized as follows: In Section 5.1, we show the $\mathcal{PLS}$-completeness of (2)-RUNCG and (2)-RDNCG. In Section 5.2, we discuss Questions 4 and 5 and also present open problems.

5.1 The Complexity of (2)-RUNCG and (2)-RDNCG

In this section, we first present an outline of our reduction from MaxCut to (2)-RDNCG and (2)-RUNCG in Subsection 5.1.1, before stepping into the construction in Subsection 5.1.2 and the proofs of correctness and tightness in Subsection 5.1.3. For our hardness result, we build on the tight $\mathcal{PLS}$-completeness of MaxCut [95]. We initially show our hardness result for directed graphs of unbounded degree and modify our construction afterwards.

The General Setting of the Reduction Let $I = (\mathcal{G}_{\mathrm{MC}} = (V_{\mathrm{MC}}, E_{\mathrm{MC}}), \mathsf{w}) \in D_{\mathrm{MC}}$ be an instance of MaxCut. Without loss of generality, we assume that $V_{\mathrm{MC}} = \{1, \ldots, n\}$. Denote

$$\mathsf{W}^\star := 2 \sum\nolimits_{e \in E} \mathsf{w}(e)$$

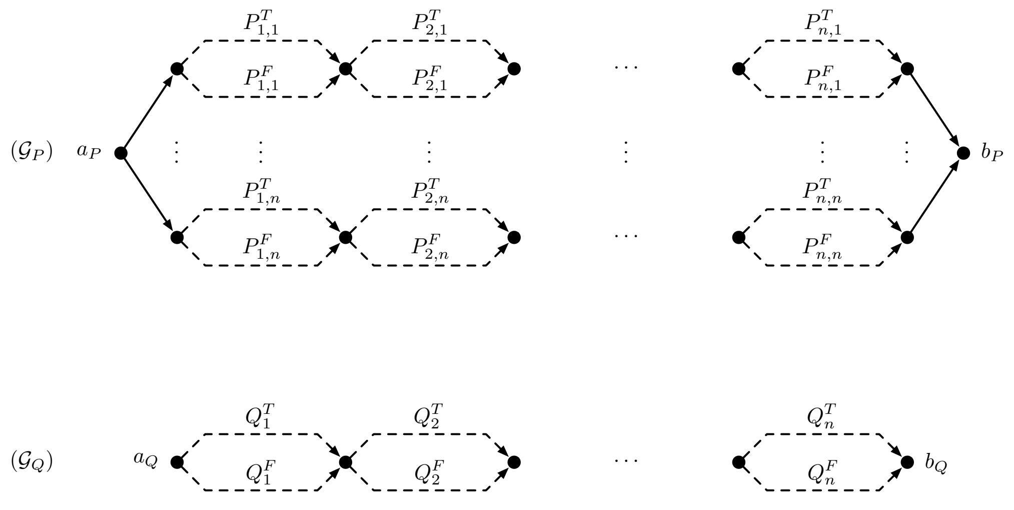


Figure 5.1: General layout of the resulting network $\mathcal{G} = (V, E)$.

and let $\mathsf{W} > \mathsf{W}^\star$. Given I, we construct an instance

$$\Phi(I) = (\mathcal{G} = (V, E), \{P, Q\}, E_P, E_Q, (a_P, b_P), (a_Q, b_Q), (d_e)_{e\in E}) \in D_{(2)\text{-RDNCG}}.$$

Here, $\mathcal{G} = (V, E)$ is a directed or undirected graph whose edges have non-decreasing delay functions $(d_e)_{e\in E}$. The two players are denoted by P and Q and have source-sink pairs (a_P, b_P) and (a_Q, b_Q). E_P denotes the set of edges player P is allowed to use and E_Q denotes the set of edges player Q is allowed to use.

Notation We use the following notation in our reduction: For $\alpha \in \{T, F\}^n$, we denote the i-th component of α by α_i. If $V = [n]$, then we denote a function $\rho : V \to \{T, F\}$ also by a vector $\rho \in \{T, F\}^n$. Since our reduction only involves two players, we denote the delay function for each edge $e \in E$ by $d_e(1)/d_e(2)$.

5.1.1 The Network and the Reduction in a Nutshell

We construct a restricted network congestion game $\mathcal{G} = (V, E)$ which consists of a subnetwork $\mathcal{G}_P = (V, E_P)$ for player P and a subnetwork $\mathcal{G}_Q = (V, E_Q)$ for player Q. Edges which can be used by both players are called *shared edges*; edges which can only be used by one player are called *exclusive edges*. Figure 5.1 depicts the network $\mathcal{G} = (V, E)$. For sake of readability, we present the underlying network paths in a compressed representation. A dashed line marked with some set of shared edges represents a subpath. We use the two types of edges to construct *two frameworks* to simulate the MaxCut-instance I in $\mathcal{G}$. The first framework consists of exclusive edges. It guarantees that by selecting strategies, each of the two players P and Q chooses a path as indicated in Figure 5.1. For player Q, a path can be described by

$\alpha \in \{T, F\}^n$ and for player P, a path can be described by (i, κ) for some $i \in [n]$ and $\kappa \in \{T, F\}^n$. A second framework consisting of some type of shared edges heavily punishes if α and κ differ by more than one bit, so that in a local optimum α and κ differ by at most one bit.

Note that α and κ can also be viewed as a partition of $V_{\mathrm{MC}} = [n]$. Our approach is rather similar to the construction used in the paper of Ackermann and Skopalik [4]. The main difference to the reduction by Ackermann and Skopalik [4], resulting in the hardness already for two players, is that players P and Q cooperational simulate the MaxCut-instance I. For each edge $e = \{u, v\} \in E_{\mathrm{MC}}$, two shared edges c_{e_1} with $e_1 = (u, v)$ and c_{e_2} with $e_2 = (v, u)$ are introduced in E. Note that u and v define the names of c_{e_1} and c_{e_2} but do not actually connect u and v in E. Our construction is such that player P chooses an assignment for u in c_{e_1} and for v in c_{e_2}; player Q chooses an assignment for v in c_{e_1} and for u in c_{e_2}. If both assignments for a pair of vertices are equal, i. e. they belong to the same partition, then both players have a delay which is equal to the weight of the edge; if both assignments disagree, then both players have no delay.

5.1.2 The Reduction

In this subsection, we present our reduction which implements the network depicted in Figure 5.1 and described in the previous subsection.

Shared Edges Our construction crucially relies on first replacing every undirected edge in the given input instance $I \in D_{\mathrm{MC}}$ by two pairs of vertices; denote the resulting set

$$E' := \{(u, v), (v, u) \mid \{u, v\} \in E\}.$$

We introduce the following shared edges:

1. For every $e \in E'$ and $i \in [n]$, we introduce two shared edges $c^T_{e,i}$ and $c^F_{e,i}$, each with delay function $0/\mathsf{w}(e)$. We call these shared edges MaxCut-*edges*. Let C denote the set of all MaxCut-edges.

2. For every $u, j \in [n]$ with $u \neq j$, we introduce two shared edges $d^T_{u,j}$ and $d^F_{u,j}$, each with delay function $0/\mathsf{W}$. We call these shared edges *deviation tester*. Let D denote the set of all deviation testers.

Exclusive Edges, Subpaths, and Strategies As outlined in the previous subsection, a dashed line marked with some set of shared edges

$$R = \{r_1, \ldots, r_k\} \subseteq C \cup D$$

represents a subpath. The subpath is described in Figure 5.2, using additional exclusive edges $s_1, s_2, \ldots, s_{k+1}$. Note that the vertices in Figure 5.1 are colored black and the newly introduced vertices in Figure 5.2 are colored white. The white vertices

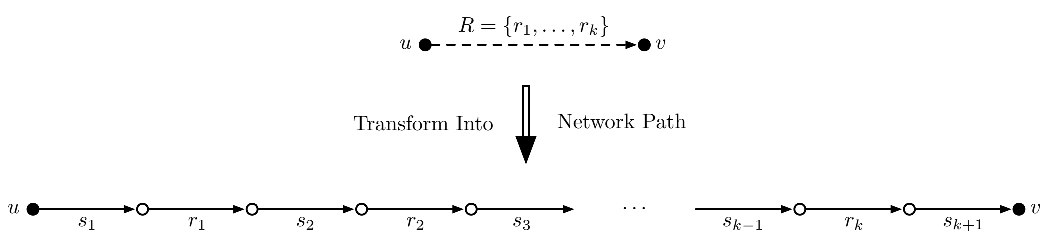


Figure 5.2: Constructing a network path between vertices u and v from its set representation.

occurring on the subpath between black vertices u, v only occur on this subpath. As indicated by Figure 5.1, the exclusive edges on a subpath defined by a set $P^{\kappa}_{u,i}$ with $\kappa \in \{T, F\}$ and $u, i \in [n]$ can only be used by player P and the exclusive edges on a subpath defined by a set Q^{α}_{u} with $\alpha \in \{T, F\}$ and $u \in [n]$ can only be used by player Q. We construct the sets $P^{\kappa}_{u,i}$ with $\kappa \in \{T, F\}$ and $u, i \in [n]$ such that they are pairwise disjoint and their union is equal to $C \cup D$. Similar, we construct the sets Q^{α}_{u} with $\alpha \in \{T, F\}$ and $u \in [n]$ such that they are pairwise disjoint and their union is equal to $C \cup D$. Let $R = \{r_1, \ldots, r_k\} \subseteq C \cup D$ define some subpath $\Pi(R)$ between black vertices u and v for, say, player P, as described above. The exclusive edges in constructing $\Pi(R)$ enforce that after entering $\Pi(R)$ through u, player P can solely traverse all shared edges in R and exit in v. Traversing less or more shared edges than present on $\Pi(R)$ would imply that player P traverses an edge which he is not allowed to use. All edges leaving a_P and all edges entering b_P can only be used by player P. So, the set of strategies for player P are then all paths connecting a_P and b_P in E_P. By construction, there is no connection between endpoints for subpaths $P^{*}_{*,i}$ and $P^{*}_{*,j}$ for all $i, j \in [n]$ with $i \neq j$. A *strategy s_P for player P* then consists of selecting some path $i \in [n]$ from the n available paths leaving a_P and selecting subpaths

$$P^{\kappa_1}_{1,i}, \ldots, P^{\kappa_n}_{n,i}$$

with $\kappa_j \in \{T, F\}$ for all $j \in [n]$; we denote this by $s_P = (i, \kappa)$. The set of strategies for player Q are all paths connecting a_Q and b_Q. A *strategy s_Q for player Q* consists of selecting subpaths

$$Q^{\alpha_1}_{1}, \ldots, Q^{\alpha_n}_{n}$$

with $\alpha_i \in \{T, F\}$ for all $i \in [n]$; we denote this by $s_Q = \alpha$.

We now present the sets of shared edges which were used to represent subpaths. For MaxCut-edges, our construction is such that for every pair of vertices $(u, v) \in E'$, player P chooses an assignment for u and player Q chooses an assignment for v. Denote $E'_1(u) := \{(u, v) \mid v \in V, (u, v) \in E'\}$. For every $\kappa \in \{T, F\}$ and $u, j \in [n]$

$$P^{\kappa}_{u,j} := \begin{cases} \{c^{\kappa}_{e,j} \mid e \in E'_1(u)\} & \text{if } u = j \\ \{d^{\kappa}_{u,j}\} \cup \{c^{\kappa}_{e,j} \mid e \in E'_1(u)\} & \text{otherwise.} \end{cases}$$

Note that subpath $P^{\kappa}_{u,u}$ does not contain any deviation tester. Denote $E'_2(u) := \{(v,u) \mid v \in V, (v,u) \in E'\}$. For every $\kappa \in \{T,F\}$ and $u \in [n]$

$$Q^{\kappa}_u := \{d^{\bar{\kappa}}_{u,j} \mid j \in [n], j \neq u\} \cup \{c^{\kappa}_{e,j} \mid e \in E'_2(u), j \in [n]\}.$$

Note that for $j \neq u$, subpaths $P^{\kappa}_{u,j}$ and $Q^{\bar{\kappa}}_u$ intersect in deviation tester $d^{\kappa}_{u,j}$, by design.

Definition 5.1. *Let* $\mathsf{s} = (s_P, s_Q) \in F(\Phi(I))$ *be a state of the game with* $s_P = (i, \kappa)$ *and* $s_Q = \alpha$*, where* $i \in [n]$ *and* $\alpha, \kappa \in \{T,F\}^n$*. Then,* s *is a* standard solution *if* $\kappa_j = \alpha_j$ *for all* $j \in [n]$ *with* $j \neq i$*.*

Solution Mapping If $\mathsf{s} \in F(\Phi(I))$ is a standard solution then $\Psi(I, \mathsf{s}) := s_Q$; if s is not a standard solution then $\Psi(I, \mathsf{s})$ returns the solution p computed by $\text{I}{\small\text{NIT}}_{\text{MC}}(I)$. This terminates the description of the reduction. ○

5.1.3 Proving the Correctness and Tightness of the Reduction

In this subsection, we show that locally optimal solutions in $\Phi(I)$ yield locally optimal solutions in I under function Ψ and that the reduction presented above is tight. Before stepping into the proof, we first present a roadmap. In the remainder of this subsection, let $\kappa, \alpha \in \{T,F\}^n$.

Roadmap of the Proof In Lemma 5.1, we show a crucial relationship between the private cost of a player in a standard solution $\mathsf{s} \in F(\Phi(I))$ and the cost of solution $\Psi(I, \mathsf{s})$ for MaxCut-instance I. Lemma 5.2 unveils that every locally optimal solution is a standard solution. Given this structural insight, we prove in Theorem 5.1 that every locally optimal solution $\mathsf{s} \in F(\Phi(I))$ induces a locally optimal solution $\Psi(I, \mathsf{s})$ for I and that our reduction is tight. Furthermore, we extend our result to hold for directed and undirected networks of degree at most three.

Lemma 5.1. *Let* $\mathsf{s} = (s_P, s_Q) \in F(\Phi(I))$ *be some state of the game with* $s_P = (i, \kappa)$ *and* $s_Q = \alpha$*. Solution* s *is a standard solution if and only if* $\delta_P(\mathsf{s}) < \mathsf{W}$ *or* $\delta_Q(\mathsf{s}) < \mathsf{W}$*. If* s *is a standard solution, then* $\delta_P(\mathsf{s}) = \delta_Q(\mathsf{s}) = \mathsf{W}^{\star} - (c_{\text{MC}}(\kappa, I) + c_{\text{MC}}(\alpha, I))$*.*

Proof. First, let s be not a standard solution. This implies that $\kappa_u = \bar{\alpha}_u$ for some $u \in [n]$ with $u \neq i$. Hence, players P and Q share deviation tester $d^{\kappa_u}_{u,i}$ with latency $0/\mathsf{W}$. This implies that $\delta_P(\mathsf{s}) \geq \mathsf{W}$ and $\delta_Q(\mathsf{s}) \geq \mathsf{W}$.

Now, let s be a standard solution and $\tau \in \{T,F\}$ in the following. Then, player P is using deviation testers $d^{\kappa_u}_{u,i}$ for all $u \in [n]$ with $u \neq i$ and player Q is using deviation testers $d^{\bar{\alpha}_u}_{u,j}$ for all $u \in [n]$ and $j \in [n]$ with $j \neq u$. Hence, every deviation tester $d^{\tau}_{u,j}$ for $u, j \in [n]$ with $u \neq j$ is used by at most one player. Furthermore, player P is using MaxCut-edges $c^{\kappa_u}_{e,i}$ for all $u \in [n]$ and $e \in E'_1(u)$; player Q is using MaxCut-edges $c^{\alpha_u}_{e,j}$ for all $u, j \in [n]$ and $e \in E'_2(u)$. Hence, for all $j \in [n]$ with $j \neq i$ and $e \in E'$, MaxCut-edge $c^{\tau}_{e,j}$ is used by at most one player. Now, consider MaxCut-edge $c^{\tau}_{e,i}$ for some $e = (u,v) \in E'$. MaxCut-edge $c^{\tau}_{e,i}$ is used by player P if and only if $\kappa_u = \tau$. MaxCut-edge $c^{\tau}_{e,i}$ is used by player Q if and only if $\alpha_v = \tau$. Since s is a standard solution, the cases $\kappa_i = \alpha_i$ and $\kappa_i \neq \alpha_i$ have to be considered:

1. First, let $\kappa = \alpha := \beta$. MaxCut-edge $c^{\tau}_{e,i}$ is used by players P and Q if and only if $\kappa_u = \alpha_v = \tau$. Hence, we obtain

$$\begin{aligned}\delta_P(\mathsf{s}) = \delta_Q(\mathsf{s}) &= 2 \sum_{\substack{e=\{u,v\}\in E\\ \beta_u=\beta_v}} \mathsf{w}(e)\\ &= \mathsf{W}^{\star} - (c_{\mathrm{MC}}(\kappa, I) + c_{\mathrm{MC}}(\alpha, I)).\end{aligned}$$

2. Now, let $\kappa \neq \alpha$. Since s is a standard solution, this implies that $\kappa_i \neq \alpha_i$ and $\kappa_j = \alpha_j := \beta_j$ for all $j \in [n]$ with $j \neq i$. Now, consider edges $e_1 = (u, v) \in E'$, $e_2 = (v, u) \in E'$ with $u, v \neq i$, and MaxCut-edges $c^{\tau}_{e_1,i}, c^{\tau}_{e_2,i}$. Then, both MaxCut-edges are used by players P and Q if and only if $\beta_u = \beta_v = \tau$. Now, let $u = i$ without loss of generality. MaxCut-edge $c^{\tau}_{e_1,i}$ is used by players P and Q if and only if $\kappa_u = \beta_v = \tau$. MaxCut-edge $c^{\tau}_{e_2,i}$ is used by players P and Q if and only if $\alpha_u = \beta_v = \tau$. Hence, we obtain

$$\begin{aligned}\delta_P(\mathsf{s}) = \delta_Q(\mathsf{s}) &= 2 \sum_{\substack{e=\{u,v\}\in E\\ u,v\neq i\\ \beta_u=\beta_v}} \mathsf{w}(e) + \sum_{\substack{\{i,v\}\in E\\ \kappa_i=\beta_v}} \mathsf{w}(e) + \sum_{\substack{\{i,v\}\in E\\ \alpha_i=\beta_v}} \mathsf{w}(e)\\ &= \mathsf{W}^{\star} - (c_{\mathrm{MC}}(\kappa, I) + c_{\mathrm{MC}}(\alpha, I)).\end{aligned}$$

□

Lemma 5.2. *Every locally optimal solution $\mathsf{s} \in F(\Phi(I))$ is a standard solution.*

Proof. Consider any state $\mathsf{s} = (s_P, s_Q) \in F(\Phi(I))$ with $s_P = (i, \kappa)$ and $s_Q = \alpha$ where $\kappa_u = \bar{\alpha}_u$ for some $u \in [n]$ with $u \neq i$. By construction, players P and Q share deviation tester $d^{\kappa_u}_{u,i}$ with latency $0/\mathsf{W}$. Then, player P can improve by switching from strategy s_P to strategy $s'_P = (i, \alpha)$. Now, $\mathsf{s}' = (s'_P, s_Q)$ is a standard solution and by Lemma 5.1, the private cost of player P in s' is strictly less that W. Hence, player P strictly decreases his private cost. Thus, s is not locally optimal.

□

Theorem 5.1. (2)-RDNCG *and* (2)-RUNCG *are tight $\mathcal{PLS}$-complete for graphs of degree at most three.*

Proof. We first show the hardness result for directed graphs of unbounded degree and modify our construction afterwards. Assume there exists a solution $\mathsf{s} = (s_P, s_Q) \in F(\Phi(I))$ with $s_P = (i, \kappa)$ and $s_Q = \alpha$, which is locally optimal for $\Phi(I)$, but $\Psi(I, \mathsf{s})$ is not locally optimal for I. By Lemma 5.2, s is a standard solution. This implies that $\kappa_k = \alpha_k$ for all $k \in [n]$ with $k \neq i$. Since $\Psi(I, \mathsf{s}) = \alpha$ is not locally optimal for I, there exists some node $v \in V$, which can be flipped such that the objective function strictly increases. Denote by $\alpha(v)$ the vector α, where the assignment of α_v is flipped, i. e. replaced by $\{T, F\} \setminus \alpha_v$. We distinguish two cases:

1. First, let $\kappa = \alpha$ or $c_{\mathrm{MC}}(\alpha(v), I) > c_{\mathrm{MC}}(\alpha(i), I)$. Then, player P has an incentive to switch to strategy $s'_P = (v, \alpha(v))$. Then, $\mathsf{s}' = (s'_P, s_Q)$ is also a standard solution and by Lemma 5.1, $\delta_P(\mathsf{s}') = \mathsf{W}^\star - (c_{\mathrm{MC}}(\alpha(v), I) + c_{\mathrm{MC}}(\alpha, I))$. Hence, player P strictly decreases his private cost.
2. Now, let $\kappa \neq \alpha$ and $c_{\mathrm{MC}}(\alpha(v), I) \leq c_{\mathrm{MC}}(\alpha(i), I)$. Then, player Q has an incentive to switch to strategy $s'_Q = \alpha(i)$. Then, $\mathsf{s}' = (s_P, s'_Q)$ is also a standard solution and by Lemma 5.1, $\delta_Q(\mathsf{s}') = \mathsf{W}^\star - (c_{\mathrm{MC}}(\kappa, I) + c_{\mathrm{MC}}(\alpha(i), I))$. Hence, player Q strictly decreases his private cost.

In both cases, we obtain a contradiction to s being locally optimal for $\Phi(I)$.

Considering tightness, we define $\mathscr{R}$ as the set of all standard solutions. By Lemma 5.2, all locally optimal solutions are contained in $\mathscr{R}$. Now, let $\mathsf{s} = (s_P, s_Q) \in \mathscr{R}$ with $s_P = (i, \kappa)$ and $s_Q = \alpha$ and let $\mathsf{s}' \in F(\Phi(I))$ be some better neighboring solution. Lemma 5.1 implies that s' is also a standard solution. We show that either $\Psi(I, \mathsf{s}) = \Psi(I, \mathsf{s}')$ or $\Psi(I, \mathsf{s}')$ is a better neighboring solution of $\Psi(I, \mathsf{s})$. If player P changes strategy in s', then $\Psi(I, \mathsf{s}) = \Psi(I, \mathsf{s}')$. If player Q changes strategy in s', then Lemma 5.1 implies that player Q switches to strategy $s'_Q = \alpha(i)$. Hence, $\Psi(I, \mathsf{s}') \in N_{\mathrm{MC}}(\Psi(I, \mathsf{s}), I)$ and by Lemma 5.1, $c_{\mathrm{MC}}(\Psi(I, \mathsf{s}), I) < c_{\mathrm{MC}}(\Psi(I, \mathsf{s}'), I)$.

Now, we show that we can adapt our construction to also work for directed graphs of degree at most three. We replace all outgoing edges from a_P by a binary tree with n leaves labeled $a_P^1, \ldots, a_P^n$. For every leaf a_P^i with $i \in [n]$, we introduce an edge (a_P^i, a_i), where a_i denotes the entry point to subpaths $P_{1,i}^T$ and $P_{1,i}^F$. A similar construction can be applied to lower the degree of node b_P to three. Furthermore, we replace all nodes connecting subpaths $P_{u,i}^\kappa$ and $P_{u,i+1}^\kappa$ for all $\kappa \in \{T, F\}$, $u \in [n]$ and $i \in [n-1]$ by a directed edge whose tail node is the exit of subpaths $P_{u,i}^\kappa$ and whose head node marks the entry for subpaths $P_{u,i+1}^\kappa$. All other nodes in our construction have degree at most three.

Finally, note that by the design of our graph, replacing all directed edges in our construction by undirected edges does not change the set of strategies for each player. Let us first have a closer look at player Q. Also in the undirected version, a simple path from a_Q to b_Q is determined by $\alpha \in \{T, F\}^n$ and follows the path marked in Figure 5.1 by $Q_1^{\alpha_1}, Q_2^{\alpha_2}, \ldots, Q_n^{\alpha_n}$. If in the undirected version, a simple path would bend after having passed $Q_1^{\alpha_1}, , \ldots, Q_i^{\alpha_i}$ for some $i \in [n]$ and go against the direction of the edges on the subpath marked by $Q_i^{\bar{\alpha}_i}$, then it would never reach b_Q. In the same way, a path from a_P to b_P in the undirected version is determined by $i \in [n]$ and $\kappa \in \{T, F\}^n$ and follows the directed path described in Figure 5.1. $\square$

5.2 Conclusion and a Discussion of Questions 4 and 5

In this chapter, we studied the complexity of computing Nash equilibria for n-player restricted network congestion games. Nash equilibria in these games coincide with local optima of Rosenthal's potential function [92], which is polynomial-time computable. This allowed to formulate the computation of a Nash equilibrium in an

n-player restricted network congestion as a $\mathcal{PLS}$ problem. The corresponding $\mathcal{PLS}$ problem for directed (resp. undirected) networks is denoted by (n)-RDNCG (resp. (n)-RUNCG).

Results Obtained For $n \geq 3$ players, the *intractability* of (n)-RDNCG was shown by Ackermann and Skopalik [4], using a $\mathcal{PLS}$ reduction from PositiveNotAll-Equal-2-Flip (a reformulation of MaxCut as a Maximum Constraint Assignment problem over binary variables; here, each constraint is a weighted clause which models an XOR of two variables). In this chapter, we showed that *already* (2)-RDNCG and (2)-RUNCG are *tight $\mathcal{PLS}$-complete* for graphs of degree at most three and hence possess the *all-exp property.* The latter implies that there exist instances and initial states of the game, such that every sequence of selfish steps has exponential length. For our constructions, we first presented the underlying idea in a nutshell in Subsection 5.1.1. The reduction from MaxCut in Subsection 5.1.2 only required two gadgets and was conceptionally simpler than the reduction presented by Ackermann and Skopalik [4]. The results we presented *are optimal* in the sense that for one player, computing a Nash equilibrium reduces to finding a shortest path, which is polynomial-time computable.

Discussion of Questions 4 and 5 The intractability results for (2)-RDNCG and (2)-RUNCG which we obtained in this chapter yield a surrounding *$\mathcal{PLS}$-completeness answer* to Question 5. Moreover, our results settle the complexity of computing a Nash equilibrium in a given restricted network congestion game. Considering Question 4, we can only provide an answer for the special case of restricted network congestion games. For restricted network congestion games, limiting the number of players to two does not significantly smoothen the complexity of computing a Nash equilibrium. Note that for *standard* congestion games with a fixed number of players, finding a Nash equilibrium is polynomial-time computable. In that case, each strategy of every player is explicitly given and therefore the total strategy space is bounded by a polynomial in the size of the input. For other models, our results do not yield an indication on the tractability of computing Nash equilibria.

Open Problems We should point out that the complexity of computing a Nash equilibrium in a *standard network congestion game* with a fixed number of players remains unsettled. Note that differing from standard congestion games, standard *network* congestion games possess a compact representation of the set of strategies of each player. Hence, the total strategy space is not bounded by a polynomial in the size of the input any longer. When using our construction as a starting point for the above open problem, an obstacle to overcome is how to transform the exclusiveness of edges for certain players to standard network congestion games. Note that our construction in Subsection 5.1.2 heavily exploited the exclusiveness of edges for certain players; in restricted network congestion games, this can be realized by definition. With this exclusiveness, we designed strategies for the two players which corresponded to, node-by-node, assignments of nodes to partitions and this eventually yielded a solution of MaxCut.

Chapter 6

On the $\mathcal{PLS}$-Complexity of Maximum Constraint Assignment

In this chapter, we investigate the complexity of computing locally optimal solutions for the fundamental $\mathcal{PLS}$ problem Maximum Constraint Assignment (in short MCA). In a nutshell, the MCA problem is a local search version of weighted Generalized Maximum Satisfiability on constraints (functions mapping assignments to positive integers) over variables with higher valence. The parameters in (p, q, r)-MCA$_{k\text{-par}}$ simultaneously limit the maximum length p of each constraint, the maximum appearance q of each variable and its valence r; additionally, the set of constraints is k-partite. We focus on *hardness* results and show $\mathcal{PLS}$-completeness of $(3, 2, 3)$-MCA$_{3\text{-par}}$ and $(2, 3, 6)$-MCA$_{2\text{-par}}$, using tight reductions from Circuit/Flip. For our results, we extend and refine a technique from Krentel [67]. The results we present are optimal in the sense that $(2, 2, r)$-MCA is solvable in polynomial time for every $r \in \mathbb{N}$. We are also interested in the special case of binary variables. For this, we investigate the (p, q, r)-VCA problem which differs from (p, q, r)-MCA in p now denoting the maximum sum of the valences of the variables minus the number of variables in each constraint. We present a general technique to reduce VCA-instances with arbitrary valence to VCA-instances over solely binary variables. As a corollary, we obtain that $(6, 2, 2)$-MCA is tight $\mathcal{PLS}$-complete.

Roadmap of this Chapter The remainder of this chapter is organized as follows: We first present an observation that all intractability results for (p, q, r)-MCA$_{k\text{-par}}$ extend to (p, q, r)-MinCA$_{k\text{-par}}$ for all $p, q, r, k \in \mathbb{N}$ in Section 6.1. Section 6.2 then presents the general method which we apply to prove Theorems 6.1 and 6.2. Section 6.3 proves the tight $\mathcal{PLS}$-completeness of $(3, 2, 3)$-MCA$_{3\text{-par}}$. Section 6.4 alters the reduction from Section 6.3 and shows the tight $\mathcal{PLS}$-completeness of $(2, 3, 6)$-MCA$_{2\text{-par}}$. Section 6.5 presents our general technique to simulate arbitrary (p, q, r)-VCA instances with with $p, q \in \mathbb{N}$ and $r \geq 3$ with VCA-instances over solely binary variables and proves Theorem 6.3 and Corollary 6.3.

6.1 On the Relation of Maximum Constraint Assignment to Minimum Constraint Assignment

The remainder of this chapter solely focuses on the MAXIMUM CONSTRAINT ASSIGNMENT problem. Let us remark that all intractability results we show in this chapter also extend to the MINIMUM CONSTRAINT ASSIGNMENT problem with the same restrictions on the set of instances, as the following lemma shows.

Lemma 6.1. *(p,q,r)-MCA$_{k\text{-}par}$ $\leq_{pls}$ (p,q,r)-MINCA$_{k\text{-}par}$ for all $p,q,r,k \in \mathbb{N}$, using a tight reduction.*

Proof. The following reduction (Φ, Ψ) only modifies the function values of the constraints in a given instance of (p,q,r)-MCA$_{k\text{-par}}$ to construct an instance of (p,q,r)-MINCA$_{k\text{-par}}$; the set of variables and the parameter list of each constraint remains untouched. Hence, where possible, we neglect the parameters $p,q,r,k \in \mathbb{N}$, for sake of readability. Let $I = (\mathcal{C}, \mathcal{X}) \in D_{\text{MCA}}$ be an instance of MAXIMUM CONSTRAINT ASSIGNMENT where each constraint has length at most p and denote

$$\mathsf{W} > \max_{C_i \in \mathcal{C}} \{C_i(\mathsf{a}(x_{i_1}), \dots, \mathsf{a}(x_{i_{p_i}})) \mid \mathsf{a}(x_{i_1}), \dots, \mathsf{a}(x_{i_{p_i}}) \in [r]^{p_i}\},$$

with $p_i \leq p$. We construct an instance $\Phi(I) = (\mathcal{C}', \mathcal{X}') \in D_{\text{MinCA}}$ of MINIMUM CONSTRAINT ASSIGNMENT, where $\mathcal{X}' := \mathcal{X}$ and for each constraint $C_i(x_{i_1}, \dots, x_{i_{p_i}}) \in \mathcal{C}$ with $p_i \leq p$, we introduce a constraint

$$C_i'(x_{i_1}, \dots, x_{i_{p_i}}) := \mathsf{W} - C_i(x_{i_1}, \dots, x_{i_{p_i}})$$

in $\mathcal{C}'$. Here, $\Psi(I, \mathsf{a}) := \mathsf{a}$ for each solution $\mathsf{a} \in F(\Phi(I))$. This terminates the description of the reduction. ○

Now, assume there exists a solution $\mathsf{a} \in F(\Phi(I))$ which is locally optimal for $\Phi(I)$, but $\Psi(I, \mathsf{a})$ is not locally optimal for I. This implies that there exists a variable $x \in \mathcal{X}$ in instance $I \in D_{\text{MCA}}$, which can be set from value $i \in [r]$ to some value $j \in [r]$ such that the objective function strictly increases by some $\Delta > 0$. By construction,

$$\begin{aligned}
\textsc{Cost}_{\text{MinCA}}(\mathsf{a}, \Phi(I)) &= \textstyle\sum_{C_i' \in \mathcal{C}'} C_i'(\mathsf{a}(x_{i_1}), \dots, \mathsf{a}(x_{i_{p_i}})) \\
&= \textstyle\sum_{C_i \in \mathcal{C}} \left(\mathsf{W} - C_i(\mathsf{a}(x_{i_1}), \dots, \mathsf{a}(x_{i_{p_i}}))\right) \\
&= |\mathcal{C}| \cdot \mathsf{W} - \textstyle\sum_{C_i \in \mathcal{C}} C_i(\mathsf{a}(x_{i_1}), \dots, \mathsf{a}(x_{i_{p_i}})) \\
&= |\mathcal{C}| \cdot \mathsf{W} - \textsc{Cost}_{\text{MCA}}(\Psi(I, \mathsf{a}), I).
\end{aligned}$$

Therefore, variable x can also be set from value $i \in [r]$ to $j \in [r]$ in $\Phi(I)$ and the objective function strictly decreases by $\Delta > 0$. When defining $\mathscr{R} := F(\Phi(I))$, it is obvious to see that our reduction is tight. □

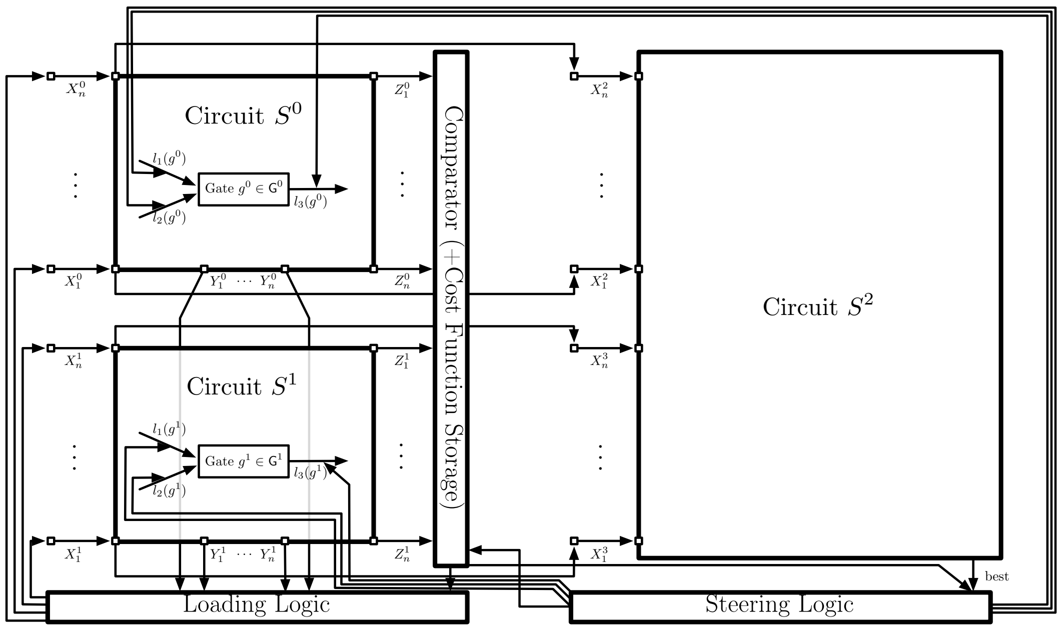


Figure 6.1: The general setting which is modeled with constraints.

6.2 The General Method for the Intractability Proofs of $(3,2,3)$-MCA$_{3\text{-par}}$ and $(2,3,6)$-MCA$_{2\text{-par}}$

In this section, we present the general method which we apply in Sections 6.3 and 6.4. We first present the setting and the general idea of our reductions before stating the necessary assumptions for Circuit/Flip. We close with the concept of propagation trees.

6.2.1 The Setting

We reduce from problem Circuit/Flip which is known to be tight $\mathcal{PLS}$-complete [56]. Given an instance $I \in D_{\text{Circuit/Flip}}$, we construct an instance of (p,q,r)-MCA$_{k\text{-par}}$. In more detail, given a circuit $\mathcal{S}$, we construct a set of constraints of length at most p from a set of variables, where each variable takes at most r values and appears in at most q constraints; additionally, the set of constraints is k-partite. Our construction extends and refines a method introduced by Krentel [67] who outlines that $(4,3,3)$-MCA is $\mathcal{PLS}$-complete. The general layout, which we model with constraints in the reductions is now as follows.

6.2.2 The Idea in a Nutshell

We depicted the general setting of our reductions in Figure 6.1. It follows the technique introduced by Krentel [67]. The reduction involves *two copies* S^0, S^1 *of the given circuit* $\mathcal{S}$, a *circuit* S^2, a *comparator*, a *loading logic*, and a *steering logic*. For all

$\kappa \in \{0,1\}$, circuit S^κ has input links $X_1^\kappa, \dots X_n^\kappa$ and computes the output bits in links $Z_1^\kappa, \dots, Z_n^\kappa$. Additional output links $Y_1^\kappa, \dots, Y_n^\kappa$ compute *the best neighboring solution* if one exists or output the input vector otherwise. While all computations in the given circuit $\mathcal{S}$ are binary, we will use values $0, 1, d$ for variables ℓ, which represent links. Binary values for ℓ model the computation of circuit $\mathcal{S}$. The additional value d (for "don't care") allows to *reset* the computation of circuits S^0 and S^1. In detail, for the reduction this means that if all variables which represent output links of a gate g are set to d, then variables which represent input links for g can be modified without changing the correctness of the constraint modeling g. The output bits from output links $Z_1^0, \dots, Z_n^0$ and $Z_1^1, \dots, Z_n^1$ are compared using a comparator that also stores the current value of the cost function. The involved constraints represent a large part of our contribution. The loading logic takes as input the result of the comparator and additionally the better neighbors $Y_1^0, \dots, Y_n^0$ and $Y_1^1, \dots, Y_n^1$. It controls loading the neighbor from the circuit whichever of the first two circuits yields the bigger output into whichever of the first two circuits yields the smaller output. Circuit S^2 has $2n$ input links labeled $X_1^2, \dots, X_n^2, X_1^3, \dots, X_n^3$ and operates on the identical input bits as circuits S^0 and S^1 in a local optimum. In the single output link best, S^2 returns the index of the input vector with the larger binary output or e if both input vectors yield the identical binary output. The results of best and the comparator feed the steering logic. The steering logic controls the reset procedure for the first two circuits and sets incentives for the special comparator. More details, outlining the control logic, which are essential for the correctness of our reductions and the improved results, are given in the respective sections.

6.2.3 Assumptions and Notation for Circuit/Flip

For an instance $I \in D_{\mathrm{C/F}}$ with circuit $\mathcal{S}$, we make the following assumptions: Without loss of generality, $c_{\mathrm{C/F}}(\mathsf{x}, I) \neq c_{\mathrm{C/F}}(\mathsf{x}', I)$ for solutions $\mathsf{x}, \mathsf{x}' \in F_{\mathrm{C/F}}(I)$ with $\mathsf{x} \neq \mathsf{x}'$. Otherwise, we may redefine the cost of each solution as

$$c'_{\mathrm{C/F}}(\mathsf{x}, I) := M \cdot c_{\mathrm{C/F}}(\mathsf{x}, I) + \mathrm{num}(\mathsf{x}).$$

Here, M is a sufficiently large integer and $\mathrm{num}(\mathsf{x})$ denotes the numerical interpretation of solution x. Circuit $\mathcal{S}$ consists of gates with at most three links. For technical reasons, every gate with three links is solely adjacent to gates with two links; every input link X_i and every output link Z_i for all $i \in [n]$ is incident to a gate with one input link and one output link. Furthermore, for every input link X_i with $i \in [n]$ and every output link Z_j with $j \in [n]$, there exists a path in circuit $\mathcal{S}$ from X_i to Z_j. For output links Z_1 and Z_n, we assume that $Z(\mathsf{x})_1 = Z(\mathsf{x})_n = 0$ for all $\mathsf{x} \in F_{\mathrm{C/F}}(I)$. We assume that for given input bits $\mathsf{x} = (x_1, \dots, x_n)$, circuit $\mathcal{S}$ additionally computes the *best* solution $\mathsf{x}' \in N_{\textsc{Circuit/Flip}}(\mathsf{x}, I)$, if such a better solution x' exists or sets $\mathsf{x}' = \mathsf{x}$ otherwise. Note that x' is polynomial-time computable and thus can be implemented in circuit $\mathcal{S}$ with at most a polynomial number of gates [69]. For representing x', *additional output links* $Y_1, \dots, Y_n$ are used, i. e. $\mathsf{x}'_i = \mathsf{R}(Y_i, \mathsf{x})$ for all $i \in [n]$.

Denote by $\textsc{Correct}_g : \{0,1\}^i \to \{T,F\}$, with $i \in \{2,3\}$ depending on gate g, the predicate describing the correct computation of the gate. We call a gate with $i \in \{1,2\}$ input links and $j \in \{1,2\}$ output links an *(i,j)-gate.* We denote the gate with input link X_i^κ for all $\kappa \in \{0,1\}$ and $i \in [n]$ by $g_{x_i^\kappa}$. For each $(1,1)$-gate $g \in \mathcal{G}$, we denote the input link by $l_1(g)$ and the output link by $l_2(g)$. For each $(2,1)$-gate $g \in \mathcal{G}$, we denote the input links by $l_1(g)$ and $l_2(g)$ and the output link by $l_3(g)$. For each $(1,2)$-gate $g \in \mathcal{G}$, we denote the input link by $l_1(g)$ and the output links by $l_2(g)$ and $l_3(g)$.

Let I be an instance of $D_{\mathrm{C/F}}$ which contains circuit $\mathcal{S}$ with n output links. Denote by S^0 and S^1 copies of circuit $\mathcal{S}$ and let $\lambda \in \{0,1\}$. Given a bit string $\mathsf{a} \in \{0,1\}^*$, where $\mathsf{a}(x^\lambda)$ denotes the assignment of input bits for S^λ, define

$$\mathrm{pos}(\mathsf{a}) := \begin{cases} \min_{i\in[n]}\{Z(\mathsf{a}(x^{\bar\lambda}))_i > Z(\mathsf{a}(x^\lambda))_i\} & \text{if } c_{\mathrm{C/F}}(\mathsf{a}(x^{\bar\lambda}), I) > c_{\mathrm{C/F}}(\mathsf{a}(x^\lambda), I) \\ n+1 & \text{otherwise.} \end{cases}$$

We drop the assignment where it is clear from the context.

6.2.4 The Concept of Propagation Trees

In our reductions, we need to spread the value of some variable a to some auxiliary variables. In each locally optimal solution, we require that if a has value α then all auxiliary variables also have value α. In both reductions, we use a *binary tree* with root a, consisting of auxiliary variables as nodes and predicates of length two as edges. Here, predicates propagate the value of the parent node u to the children v_1, v_2. The actual implementation in the two reductions now differs in the use of the predicates. In our reduction for $(2,3,*)$-MCA$_{2\text{-par}}$, we include these predicates as constraints to spread the value of a; by construction, every such constraint has length two and every variable appears at most three times. In our reduction for $(3,2,*)$-MCA$_{3\text{-par}}$, we introduce a constraint consisting of the predicates for $\{u, v_1\}$ and $\{u, v_2\}$. By construction, every such constraint has length three and every variable appears twice. Generally, weights exponentially decrease on each level from the root node to the leaves. We denote the root of some propagation tree T by $\mathrm{root}(T)$ and its set of leaves by $\mathrm{leaves}(T)$.

6.3 $(3,2,3)$-MCA$_{3\text{-par}}$ is Tight $\mathcal{PLS}$-Complete

In this section, we show that $(3,2,3)$-MCA$_{3\text{-par}}$ is tight $\mathcal{PLS}$-complete. The variables, predicates, and constraints are defined with their extensibility in mind. We present the reduction function Φ and the solution mapping Ψ. Given an instance $I \in D_{\textsc{Circuit/Flip}}$, we construct an instance

$$\Phi(I) = (\mathcal{F}, \mathcal{X}) \in D_{(3,2,3)\text{-MCA}_{3\text{-par}}}$$

with a set of constraints $\mathcal{F}$ and a set of at most ternary variables $\mathcal{X}$. Every constraint $C_i \in \mathcal{F}$ has length at most 3 and every variable $x \in \mathcal{X}$ appears in at most 2 constraints.

Identifier	Meaning
S^i	copy i of input circuit $\mathcal{S}$
$\mathcal{G}^i$	set of gates in circuit S^i
$\mathcal{G}^i_{\text{in}}$	set of gates incident to some input link in circuit S^i
$\mathcal{L}^i$	set of links in circuit S^i
$\mathcal{X}^i_{\mathcal{L}}$	set of variables corresponding to links in $\mathcal{L}^i$
$\tilde{\mathcal{Q}}$	$\mathcal{Q}^0 \cup \mathcal{Q}^1$, for all $\mathcal{Q} \in \{\mathcal{G}, \mathcal{L}, \mathcal{X}_{\mathcal{L}}\}$

Table 6.1: The labeling and meaning of identifiers we use. Here, $i \in \{0, 1, 2\}$.

6.3.1 The Set of Constraints

In our reduction, we use the notation for sets as listed in Table 6.1. We define constraints which are composed of sets of predicates. In the following, whenever we use κ in the index of a predicate this is to denote that there are actually two predicates in $\mathcal{F}$, one for $\kappa = 0$ and one for $\kappa = 1$. Constraints consisting of more than one weighted predicate are listed in Table 6.2. All other constraints are weighted predicates. The *set of constraints* is then

$$\begin{aligned}\mathcal{F} := \tilde{\mathcal{G}} \cup \mathcal{G}^2 \cup \{D_i, P_i, \mathsf{B}^{\kappa}_{\text{load}_i} \mid i \in [n]\} \cup \\ \{Q_i \mid i \in [n-1]\} \cup \{\mathsf{B}_{\text{propagate}}, \mathsf{B}_{\text{setIncentives}}\} \cup T,\end{aligned}$$

where T denotes the set of predicates for all propagation trees T_j with $j \in [3]$ (see next paragraph for details). We make the set of constraints hierarchical by defining the weights of the predicates in powers of 2. For joining expressions within a predicate, we write "|" to denote the boolean `OR` operator. We assign numbers (6.1), . . . , (6.23) to the predicates and predicates with a higher number have smaller weight. Within every label that contains more than one predicate, weights increase or decrease.

6.3.2 The Set of Variables

We will use the same letters for links and variables. With slight abuse of notation, we denote the variables and gates for all special links X^{κ}_i, $X^{\kappa+2}_i$, Y^{κ}_i, and Z^{κ}_i with $i \in [n]$ and $\kappa \in \{0, 1\}$ in small letters. Recall that by assumption on input circuit $\mathcal{S}$, each gate $g \in \tilde{\mathcal{G}}_{\text{in}}$ is a $(1, 1)$-gate. We introduce the *link variables* listed in the upper section of Table 6.3. We additionally introduce link variables with special names y^{κ}_i and z^{κ}_i in $\mathcal{X}^{\kappa}_{\mathcal{L}}$ and $x^{\kappa+2}_i$ in $\mathcal{X}^2_{\mathcal{L}}$ for all $i \in [n]$. We call a variable associated to an input link of a gate $g \in \mathcal{G}$ *input variable of* g, correspondingly for output links. We introduce the *variables for comparison and controlling* listed in the middle and lower section of Table 6.3; for sake of readability, we omit identical comments in succeeding rows. Here, propagation tree T_1 spreads the value of a to all a^0_i, a^1_i for $i \in [n]$; propagation tree T_2 spreads the value of b to b' and all b_g for $g \in \tilde{\mathcal{G}}$ a $(1, 1)$-gate; propagation tree T_3 spreads the value of c to all variables c_i for $i \in [0 : n-1]$. Eventually, the *set of*

Constraint,	Index	Set of weighted predicates	Identifiers
D_i,	$i \in [n]$	$\mathsf{A}_{\mathrm{corr}(D_i)}, \mathsf{A}_{\mathrm{out}(D_i)}$	(6.2), (6.6)
P_1		$\mathsf{A}^3_{\mathrm{carry}(P_1)}, \mathsf{A}^1_{\mathrm{care}(P_i)}, \mathsf{C}_{\mathrm{incentive}(P_1)}$	(6.7), (6.10), (6.21)
P_i,	$i \in [2 : n-2]$	$\mathsf{A}^1_{\mathrm{carry}(P_i)}, \mathsf{A}^2_{\mathrm{carry}(P_i)}, \mathsf{A}^3_{\mathrm{carry}(P_i)}, \mathsf{A}^0_{\mathrm{care}(P_i)}, \mathsf{A}^1_{\mathrm{care}(P_i)}, \mathsf{C}_{\mathrm{incentive}(P_i)}$	(6.4), (6.5), (6.7), (6.8), (6.10), (6.22)
P_{n-1}		$\mathsf{A}^1_{\mathrm{carry}(P_{n-1})}, \mathsf{A}^2_{\mathrm{carry}(P_{n-1})}, \mathsf{A}^3_{\mathrm{carry}(P_{n-1})}, \mathsf{C}_{\mathrm{incentive}(P_{n-1})}$	(6.4), (6.5), (6.7), (6.22)
P_n		$\mathsf{A}^1_{\mathrm{carry}(P_n)}, \mathsf{A}^2_{\mathrm{carry}(P_n)}, \mathsf{A}^3_{\mathrm{carry}(P_n)}$	(6.4), (6.5), (6.7)
Q_i,	$i \in [2 : n-1]$	$\mathsf{A}^0_{\mathrm{carry}(Q_i)}, \mathsf{A}_{\mathrm{care}(Q_i)}, \mathsf{C}^\kappa_{\mathrm{incentive}(Q_i)}$	(6.3), (6.9), (6.19), (6.20)
Q_n		$\mathsf{A}^0_{\mathrm{carry}(Q_n)}, \mathsf{C}^0_{\mathrm{incentive}(Q_n)}$	(6.3), (6.20)
g,	$g \in \tilde{\mathcal{G}}_{\mathrm{in}}$	$\mathsf{A}_{\mathrm{corr}(g)}, \mathsf{B}_{\mathrm{copy}(g)}$	(6.1), (6.13)
g,	$g \in \tilde{\mathcal{G}} \setminus \tilde{\mathcal{G}}_{\mathrm{in}}$, a $(1,1)$-gate	$\mathsf{A}_{\mathrm{corr}(g)}, \mathsf{C}^{\kappa+1}_{\mathrm{value}(g)}$	(6.1), (6.23)
g,	$g \in \tilde{\mathcal{G}}$, a $(1,2)$-gate	$\mathsf{A}^\kappa_{\mathrm{corr}(g)}$	(6.1), (6.1)

Table 6.2: Constraints consisting of more than one weighted predicate for $\kappa \in \{0,1\}$.

Variable	Index	Domain	Constraint	Comment
l	$l \in \tilde{\mathcal{X}}_{\mathcal{L}} \setminus \{x_i^\kappa, y_i^\kappa, z_i^\kappa \mid i \in [n]\}$	$\{0,1,d\}$	g, g'	Variables for links in S^0, S^1
x_i^κ	$i \in [n]$	$\{0,1\}$	$g, \mathsf{B}^{\bar\kappa}_{\mathrm{load}_i}$	Special variables for links X_i^κ
y_i^κ	$i \in [n]$	$\{0,1,d\}$	$g,\ \mathsf{B}^{\kappa}_{\mathrm{load}_i}$	Special variables for links Y_i^κ
z_i^κ	$i \in [n]$	$\{0,1,d\}$	$g,\ D_i$	Special variables for links Z_i^κ
l	$l \in \mathcal{X}^2_{\mathcal{L}} \setminus \{\{\text{best}\} \cup \{x_i^{\kappa+2} \mid i \in [n]\}$	$\{0,1\}$	g, g'	Variables for links in S^2
$x_i^{\kappa+2}$	$i \in [n]$	$\{0,1\}$	$g,\ \mathsf{B}^{\bar\kappa}_{\mathrm{load}_i}$	Special variables for links X_i^κ
best		$\{0,1,e\}$	$g,\ \mathsf{B}_{\mathrm{propagate}}$	Special variable for link best
comp_i	$i \in [n-1]$	$\{0,1,e\}$	D_i, P_i	Ternary comparator variables
$\mathrm{comp}^\star_{2i}$	$i \in [n-2]$	$\{0,1,e\}$	P_i, Q_{i+1}	
$\mathrm{comp}^\star_{2i+1}$	$i \in [n-2]$	$\{0,1,e\}$	Q_{i+1}, P_{i+1}	
comp_n		$\{0,1\}$	D_n, P_n	Binary comparator variables
$\mathrm{comp}^\star_{2n-2}$		$\{0,1\}$	P_{n-1}, Q_{n-1}	
$\mathrm{comp}^\star_{2n-1}$		$\{0,1\}$	Q_n, P_n	
$\mathrm{comp}^\star_{2n}$		$\{0,1\}$	$P_n, N^1(\mathrm{comp}^\star_{2n})$	
a		$\{0,1\}$	$N^1(a),\ \mathsf{B}_{\mathrm{propagate}}$	Control variables
a'		$\{0,1\}$	$N^1(a'),\ \mathsf{B}_{\mathrm{setIncentives}}$	
a_i^κ	$i \in [n]$	$\{0,1\}$	$N^1(a_i^\kappa), \mathsf{B}^{\bar\kappa}_{\mathrm{load}_i}$	
b		$\{0,1\}$	$\mathsf{B}_{\mathrm{propagate}}, N^2(b)$	
b'		$\{0,1\}$	$N^2(b'), \mathsf{B}_{\mathrm{setIncentives}}$	
b_g	$g \in \tilde{\mathcal{G}}$, a $(1,1)$-gate	$\{0,1\}$	$N^2(b_g), g$	
c		$\{0,1,e\}$	$N^3(c), P_1$	
c_i	$i \in [0:n-1]$	$\{0,1,e\}$	$N^3(c_i), Q_i$	

Table 6.3: The set of variables, their respective domains and constraints for $\kappa \in \{0,1\}$. Here, constraints g and g' refer to gates with the respective input or output variable; $N^i(x)$ denotes the constraint containing variable $x \in \mathcal{X}$ in propagation tree T_i with $i \in [3]$.

variables is

$$\begin{aligned}\mathcal{X} := \tilde{\mathcal{X}}_{\mathcal{L}} \cup \mathcal{X}^2_{\mathcal{L}} \cup \{\text{comp}_i \mid i \in [n]\} \cup \{\text{comp}^\star_i \mid i \in [2:2n]\} \cup \\ \{a^0_i, a^1_i, c_{i-1} \mid i \in [n]\} \cup \{b_g \mid g \in \tilde{\mathcal{G}},\ \text{a } (1,1)\text{-gate}\} \cup \\ \{a, a', b, b', c, \text{best}\} \cup V_{T_1} \cup V_{T_2} \cup V_{T_3},\end{aligned}$$

where V_{T_i} with $i \in [3]$ denotes the set of variables in propagation tree T_i.

Our Notation for Values of Variables

Depending on the context, we use three different notations to denote the value of a variable $x \in \mathcal{X}$: If the solution $\mathsf{a} \in F(\Phi(I))$ is relevant, then we denote the assignment of variable x by $\mathsf{a}(x)$. For sake of readability, we drop the solution, where it is clear from the context and then use the following notation: For all link and control variables ℓ, including variable best, but excluding special link variables for input and output links, we denote the assignment by $v(\ell)$. For all other variables, we use the name and the value of a variable interchangeable.

6.3.3 The Constraint-Graph of Our Reduction

Since every variable appears in at most two constraints, we can interpret an instance of $(3,2,3)$-MCA$_{3\text{-par}}$ as a *graph*, where constraints are nodes and variables are edges. Figure 6.2 shows the resulting graph of our reduction. Boxes represent constraints containing the predicates sorted by weight in decreasing order. For sake of readability, we additionally included the names of the variables spanning edges in the graph. Furthermore, we used thinner lines for edges spanned by auxiliary variables from propagation trees.

6.3.4 A More Detailed Overview of the Reduction

Figure 6.2 also depicts a more detailed overview of the reduction where boxes are constraints and arcs are variables. The variables control different tasks, according to the general setting outlined in Section 6.2.

The computation on *three copies* of input circuit $\mathcal{S}$ is simulated. We use variables in $\tilde{\mathcal{X}}_{\mathcal{L}}$ to simulate circuits S^0 and S^1. Here, binary values 0 and 1 simulate the computation of the corresponding circuit and the auxiliary value d allows to reset the computation. Our construction is such that if in a locally optimal solution $v(\ell) = d$ for some link variable $\ell \in \tilde{\mathcal{X}}_{\mathcal{L}} \setminus \{x^\kappa_1, \ldots, x^\kappa_n\}$, then $v(\ell_s) = d$ holds also for all link variables ℓ_s which are successors of ℓ in the topological order of $\mathcal{S}$. We model circuit S^2 with binary variables from $\mathcal{X}^2_{\mathcal{L}}$. In a locally optimal solution, circuit S^2 operates on the same input bits as circuits S^0 and S^1. If all predicates for gates in circuit S^2 are correct, then the computation—and in particular the value of variable best—is correct, with respect to the input. This differs from circuits S^0 and S^1, where all predicates for gates may be correct, but all output variables are d and are thus of little help in determining the circuit with the larger output. Recall that by assumption,

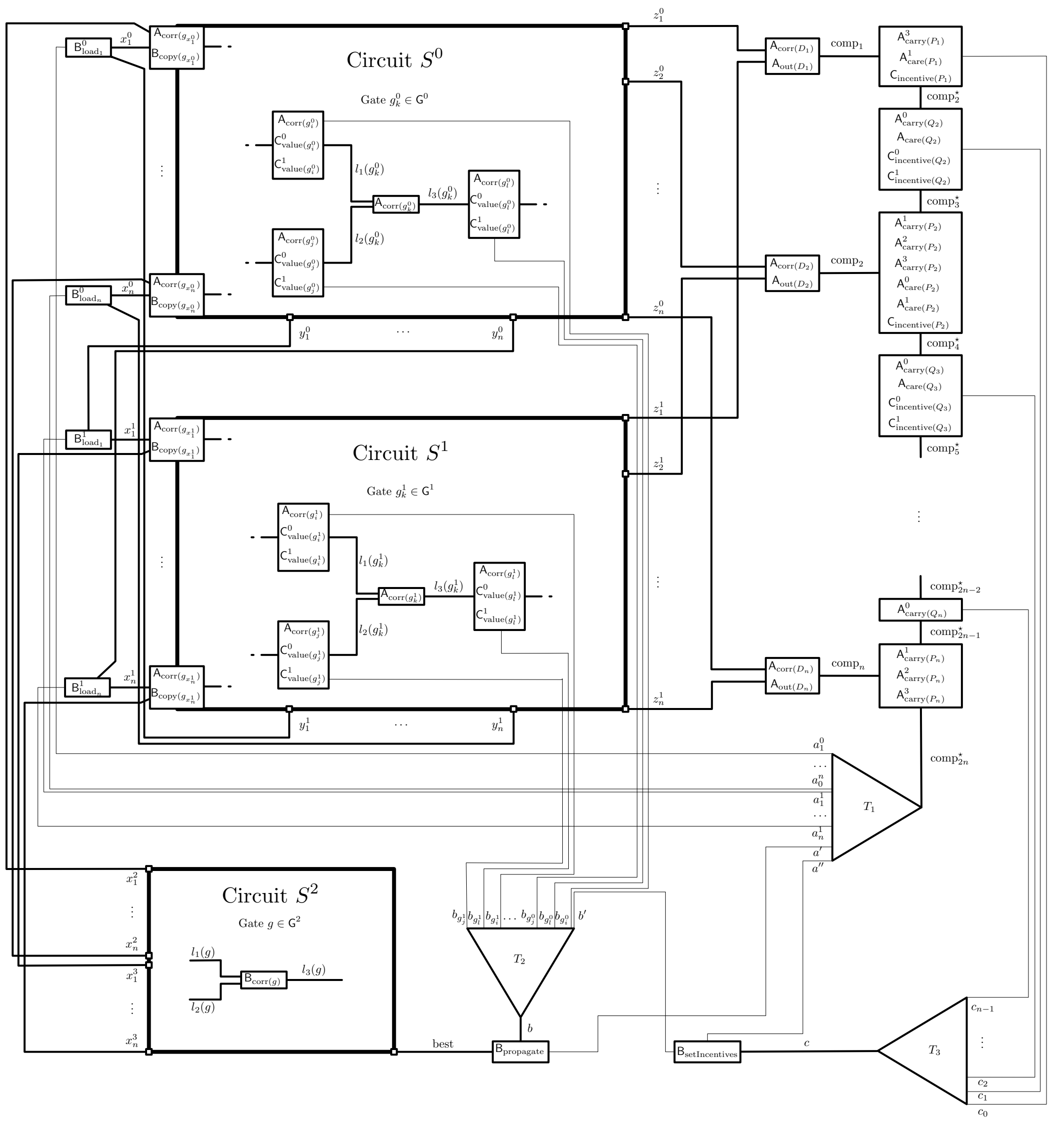

Circuit S^0
Gate $g_k^0 \in \mathsf{G}^0$
Circuit S^1
Gate $g_k^1 \in \mathsf{G}^1$
Circuit S^2
Gate $g \in \mathsf{G}^2$
$\mathsf{B}^0_{\text{load}_1}$
$\mathsf{B}^0_{\text{load}_n}$
$\mathsf{B}^1_{\text{load}_1}$
$\mathsf{B}^1_{\text{load}_n}$
T_1
T_2
T_3
$\mathsf{B}_{\text{propagate}}$
$\mathsf{B}_{\text{setIncentives}}$
best
b
c
comp_1
comp_2
comp_n
$\text{comp}^\star_{2n}$

Figure 6.2: The constraint-graph of our reduction. Boxes are constraints and arcs are variables.

circuit $\mathcal{S}$ outputs the best neighbor in case one exists or the input bits otherwise. In our construction, the new input bits for circuit S^κ with $\kappa \in \{0,1\}$ are either the input bits of circuit $S^{\bar\kappa}$ if no better neighbor exists for input bits of circuit $S^{\bar\kappa}$ or the best neighbor for input bits of circuit $S^{\bar\kappa}$.

The *core of our reduction* are predicates (6.2)–(6.7) which model the comparator. Compared to Krentel [67], we change these predicates and this is a key modification in order to lower the maximum length of any constraint. Additionally, this leads to a more focused control structure and improves the readability. Predicates (6.2)–(6.7) control the setting of the *weighted output*, when an improved solution is found. For these constraints, variables, comp_i, $\text{comp}^\star_{2i}$, and $\text{comp}^\star_{2i+1}$ for $i \in [n]$ are used. In a locally optimal solution, the values of all variables modeling the comparator point to the index of the circuit with the larger output. Variables in V_{T_1} for propagation tree T_1 forward the result of the comparator to the loading logic for which we use variables $a_{\dots}$. The steering logic consists of propagation trees T_2 and T_3 and its values are largely controlled by output link best of circuit S^2. Variables $b_{\dots}$ in V_{T_2} for propagation tree T_2 support loading of a better neighbor. Variables $c_{\dots}$ in V_{T_3} for propagation tree T_3 set the small incentives for the comparator in dependence of its current value and the value of best. We define the weights of the predicates in the following decreasing order: predicates controlling the computation of circuits S^0 and S^1 are the heaviest, followed by predicates modeling the comparator. Predicates which model the loading logic outweigh the corresponding predicates for circuit S^2, which are again larger than the corresponding predicates for the steering logic.

In order to show tightness of our construction, we introduce crucial additional predicates (6.8)–(6.10). These predicates trim the possibilities for changes to variables modeling the comparator in each improving step and are carefully designed not to interfere with the correctness of our construction. Let us stress that for the proof of $\mathcal{PLS}$-completeness, predicates (6.8)–(6.10) are not necessary.

6.3.5 The Set of Predicates

In this subsection, we present a complete description of the set of constraints. Recall that whenever we use κ in the index of a predicate this is to denote that there are actually two predicates in $\mathcal{F}$, one for $\kappa = 0$ and one for $\kappa = 1$. We introduce predicates in the following three levels:

Level I: Computing and Comparing the Output

The heaviest predicates model the computation of gates. All $(2,1)$-gates $g \in \tilde{\mathcal{G}}$ are described by predicates of *length three*

$$\begin{aligned}&\mathsf{A}_{\text{corr}(g)}(l_1(g), l_2(g), l_3(g)) = \\ &\quad [(l_1(g) = d \vee l_2(g) = d) \Rightarrow l_3(g) = d] \text{ and} \\ &\quad [l_1(g), l_2(g), l_3(g) \in \{0,1\} \Rightarrow \textsc{Correct}_g(l_1(g), l_2(g), l_3(g))].\end{aligned} \tag{6.1}$$

Note that $\mathsf{A}_{\mathrm{corr}(g)}$ is satisfied, if $l_3(g) = d$. All $(1,2)$-gates $g \in \tilde{\mathcal{G}}$ are described by two predicates

$$\begin{aligned}\mathsf{A}^0_{\mathrm{corr}(g)}(l_1(g), l_2(g)) = [&l_2(g) = d \mid \\ &l_1(g), l_2(g) \in \{0,1\} \wedge \textsc{Correct}_g(l_1(g), l_2(g))].\end{aligned} \tag{6.1}$$

$$\begin{aligned}\mathsf{A}^1_{\mathrm{corr}(g)}(l_1(g), l_3(g)) = [&l_3(g) = d \mid \\ &l_1(g), l_3(g) \in \{0,1\} \wedge \textsc{Correct}_g(l_1(g), l_3(g))].\end{aligned} \tag{6.1}$$

All $(1,1)$-gates $g \in \tilde{\mathcal{G}}$ are described by predicate $\mathsf{A}_{\mathrm{corr}(g)}(l_1(g), l_2(g))$ which is defined as predicate $\mathsf{A}^0_{\mathrm{corr}(g)}(l_1(g), l_2(g))$ in (6.1). The weights of the predicates decrease in the topological order of the gates. For each satisfied predicate, values 0 and 1 mean that with respect to gate g, the output variable is verified to be correct and d (for "don't care") means that the output variable is unverified. The core of our reduction are the following predicates (6.2)–(6.7). For all $i \in [n]$, we introduce predicates for the comparison in

$$\begin{aligned}&\mathsf{A}_{\mathrm{corr}(D_i)}(\mathrm{comp}_i, z_i^0, z_i^1) = \\ &\quad [\mathrm{comp}_i = e \wedge z_i^0 = z_i^1 \in \{0,1\} \wedge i < n \mid \\ &\quad\ \mathrm{comp}_i = 0 \wedge z_i^0 \in \{0,1\} \mid \\ &\quad\ \mathrm{comp}_i = 1 \wedge z_i^1 \in \{0,1\}].\end{aligned} \tag{6.2}$$

Predicates (6.3)–(6.7) for the *comparator* are in interleaving order $\dots$, $\mathsf{A}^0_{\mathrm{carry}(Q_i)}$, $\mathsf{A}^1_{\mathrm{carry}(P_i)}$, $\mathsf{A}^2_{\mathrm{carry}(P_i)}$ $\mathsf{A}_{\mathrm{out}(D_i)}$, $\mathsf{A}^3_{\mathrm{carry}(P_i)}$, $\mathsf{A}^0_{\mathrm{carry}(Q_{i+1})}$, $\mathsf{A}^1_{\mathrm{carry}(P_{i+1})}$, $\mathsf{A}^2_{\mathrm{carry}(P_{i+1})}$, $\mathsf{A}_{\mathrm{out}(D_{i+1})}$, $\mathsf{A}^3_{\mathrm{carry}(P_{i+1})}$, $\dots$ for all $i \in [n-1]$ and weights decrease in this order; for $i = 1$, predicates (6.3)–(6.5) are not defined. Recall that variables comp_n, $\mathrm{comp}^\star_{2n}$ and $\mathrm{comp}^\star_{2n-1}$ are defined as binary variables. For all $i \in [2:n]$, we introduce predicates which propagate the values from $\mathrm{comp}^\star_{2i-2}$ to $\mathrm{comp}^\star_{2i-1}$ in

$$\begin{aligned}\mathsf{A}^0_{\mathrm{carry}(Q_i)}(\mathrm{comp}^\star_{2i-2}, \mathrm{comp}^\star_{2i-1}) = [&\mathrm{comp}^\star_{2i-2} = e \wedge i < n \mid \\ &\mathrm{comp}^\star_{2i-2} = \mathrm{comp}^\star_{2i-1} \in \{0,1\}].\end{aligned} \tag{6.3}$$

For all $i \in [2:n]$, we introduce predicates which propagate the values from $\mathrm{comp}^\star_{2i-1}$ to $\mathrm{comp}^\star_{2i}$ in predicate

$$\begin{aligned}\mathsf{A}^1_{\mathrm{carry}(P_i)}(\mathrm{comp}^\star_{2i-1}, \mathrm{comp}^\star_{2i}) = [&\mathrm{comp}^\star_{2i-1} = e \wedge i < n \mid \\ &\mathrm{comp}^\star_{2i-1} = \mathrm{comp}^\star_{2i} \in \{0,1\}].\end{aligned} \tag{6.4}$$

For all $i \in [2:n]$, we use a special predicate to propagate the value of $\mathrm{comp}^\star_{2i-1}$ to comp_i in

$$\begin{aligned}\mathsf{A}^2_{\mathrm{carry}(P_i)}(\mathrm{comp}^\star_{2i-1}, \mathrm{comp}_i) = [&\mathrm{comp}^\star_{2i-1} = e \wedge i < n \mid \\ &\mathrm{comp}_i = e \wedge i < n \mid \\ &\mathrm{comp}^\star_{2i-1} = \mathrm{comp}_i \in \{0,1\}].\end{aligned} \tag{6.5}$$

For all $i \in [n]$, we define predicates rewarding output

$$\begin{aligned}\mathsf{A}_{\text{out}(D_i)}(\text{comp}_i, z_i^0, z_i^1) = &\\ &[\text{comp}_i = e \wedge z_i^0 = z_i^1 = 1 \wedge i < n \mid \\ &\ \text{comp}_i = 0 \wedge z_i^0 = 1 \mid \\ &\ \text{comp}_i = 1 \wedge z_i^1 = 1].\end{aligned} \tag{6.6}$$

For all $i \in [n]$, we introduce predicates propagating the ternary values from comp_i to $\text{comp}^\star_{2i}$ in

$$\begin{aligned}\mathsf{A}^3_{\text{carry}(P_i)}(\text{comp}_i, \text{comp}^\star_{2i}) = [&\text{comp}_i = e \wedge i < n \mid \\ &\text{comp}_i = \text{comp}^\star_{2i} \in \{0,1\}].\end{aligned} \tag{6.7}$$

For every $i \in [2:n-2]$, $j \in [2:n-1]$, and $k \in [n-2]$, we introduce additional predicates in

$$\mathsf{A}^0_{\text{care}(P_i)}(\text{comp}^\star_{2i-1}, \text{comp}^\star_{2i}) = [\text{comp}^\star_{2i-1} = e \vee \text{comp}^\star_{2i} \in \{0,1\}] \tag{6.8}$$

$$\mathsf{A}_{\text{care}(Q_j)}(\text{comp}^\star_{2j-2}, \text{comp}^\star_{2j-1}) = [\text{comp}^\star_{2j-2} = e \vee \text{comp}^\star_{2j-1} \in \{0,1\}]. \tag{6.9}$$

$$\mathsf{A}^1_{\text{care}(P_k)}(\text{comp}_k, \text{comp}^\star_{2k}) = [\text{comp}_k = e \vee \text{comp}^\star_{2k} \in \{0,1\}] \tag{6.10}$$

Recall that variables comp_n and $\text{comp}^\star_j$ for all $j \in [2n-2:2n]$ are defined as binary variables. The value of $\text{comp}^\star_{2n}$ is propagated to all variables a_i^κ with $i \in [n]$ and a, a', using propagation tree T_1 where

$$\begin{aligned}\text{root}(T_1) &= \text{comp}^\star_{2n};\\ \text{leaves}(T_1) &= \{a, a'\} \cup \{a_i^\kappa \mid i \in [n], \kappa \in \{0,1\}\}.\end{aligned} \tag{6.11}$$

Level II: Loading a Better Neighbor and Pushing the Comparator

For all $i \in [n]$, if $a_i^{\bar\kappa} = \kappa$ then we copy the values of y_i^κ to $x_i^{\bar\kappa}$ in predicates

$$\mathsf{B}^\kappa_{\text{load}_i}(x_i^{\bar\kappa}, y_i^\kappa, a_i^{\bar\kappa}) = [x_i^{\bar\kappa} = y_i^\kappa \wedge a_i^{\bar\kappa} = \kappa]. \tag{6.12}$$

The weights are decreasing in i. For all x_i^κ with $i \in [n]$, we copy the values of x_i^κ to $x_i^{\kappa+2}$ in predicates

$$\mathsf{B}_{\text{copy}(g_{x_i^\kappa})}(x_i^\kappa, x_i^{\kappa+2}) = [x_i^\kappa = x_i^{\kappa+2}]. \tag{6.13}$$

For all $(2,1)$-gates $g \in \mathcal{G}^2$, we introduce predicates

$$\mathsf{B}_{\text{corr}(g)}(l_1(g), l_2(g), l_3(g)) = [\textsc{Correct}_g(l_1(g), l_2(g), l_3(g))]. \tag{6.14}$$

We use a similar predicate for $(1,2)$-gates $g \in \mathcal{G}^2$ with the modification that $\mathsf{B}_{\text{corr}(g)}(l_1(g), l_2(g))$ splits into two predicates; predicate $\mathsf{B}^0_{\text{corr}(g)}(l_1(g), l_3(g))$ describes the dependence of $l_2(g)$ on $l_1(g)$ and predicate $\mathsf{B}^1_{\text{corr}(g)}$ describes the dependence of

$l_3(g)$ on $l_1(g)$. All $(1,1)$-gates $g \in \mathcal{G}^2$ are described by predicate $\mathsf{B}_{\text{corr}(g)}(l_1(g), l_2(g))$ which is defined as predicate $\mathsf{B}^0_{\text{corr}(g)}(l_1(g), l_2(g))$. The weights of the predicates again decrease in the topological order of the gates. Recall that in the single output link best, S^2 returns the index of the input vector which yields the larger binary output or e if both input vectors yield the identical binary output. If best $\neq e$ then its value is propagated to variable b, otherwise the value of a is copied to variable b in predicate

$$\begin{aligned}\mathsf{B}_{\text{propagate}}(\text{best}, a, b) = \\ &[\text{best} = e \wedge b = a \mid \\ &b = \text{best}].\end{aligned} \tag{6.15}$$

The value of b is then propagated to variable b' and all b_g with $g \in \tilde{\mathcal{G}}$ a $(1,1)$-gate, using propagation tree T_2, where

$$\begin{aligned}\text{root}(T_2) &= b;\\ \text{leaves}(T_2) &= \{b'\} \cup \{b_g \mid g \in \tilde{\mathcal{G}}, \text{ a } (1,1)\text{-gate}\}.\end{aligned} \tag{6.16}$$

We determine the value of variable c in predicate

$$\begin{aligned}\mathsf{B}_{\text{setIncentives}}(b', a', c) = \\ &[a' \neq b' \wedge v(c) = e \mid \\ &a' = b' \wedge v(c) = b'].\end{aligned} \tag{6.17}$$

The value of c is then propagated to all variables c_i with $i \in [0 : n-1]$, using propagation tree T_3, where

$$\begin{aligned}\text{root}(T_3) &= c;\\ \text{leaves}(T_3) &= \{c_i \mid i \in [0 : n-1]\}.\end{aligned} \tag{6.18}$$

Level III: Small Incentives

For all $i \in [2 : n]$ and $j \in [2 : n-1]$, we introduce predicates which reward setting all $\text{comp}^\star_{2i-2}$, $\text{comp}^\star_{2i-1}$ to c_{i-1} and all $\text{comp}^\star_{2j-2}$, $\text{comp}^\star_{2j-1}$ to c_{j-1} in

$$\mathsf{C}^0_{\text{incentive}(Q_i)}(c_{i-1}, \text{comp}^\star_{2i-2}) = [c_{i-1} = \text{comp}^\star_{2i-2}] \tag{6.19}$$

$$\mathsf{C}^1_{\text{incentive}(Q_j)}(c_{j-1}, \text{comp}^\star_{2j-1}) = [c_{j-1} = \text{comp}^\star_{2j-1}]. \tag{6.20}$$

The weights are increasing in i. We introduce a predicate which rewards setting comp_1 to c_0 in

$$\mathsf{C}_{\text{incentive}(P_1)}(c_0, \text{comp}_1) = [c_0 = \text{comp}_1]. \tag{6.21}$$

For all $i \in [2 : n-1]$, we introduce an incentive to set comp_i according to $\text{comp}^\star_{2i-1}$ in

$$\mathsf{C}_{\text{incentive}(P_i)}(\text{comp}^\star_{2i-1}, \text{comp}_i) = [\text{comp}^\star_{2i-1} = \text{comp}_i]. \tag{6.22}$$

For every $(1,1)$-gate $g^\kappa \in \tilde{\mathcal{G}} \setminus \tilde{\mathcal{G}}_{\text{in}}$, we introduce small incentives for its link variables $l_1(g^\kappa), l_2(g^\kappa)$ for all $i = 1, 2$ in

$$\begin{aligned}\mathsf{C}^i_{\text{value}(g^\kappa)}&(b_{g^\kappa}, l_i(g^\kappa)) = \\ &[b_{g^\kappa} = \bar{\kappa} \wedge l_i(g^\kappa) = d \mid \\ &\ b_{g^\kappa} = \kappa \wedge l_i(g^\kappa) \in \{0,1\}].\end{aligned} \tag{6.23}$$

The weights of the predicates increase in the topological order of the links. Recall that by assumption on $\mathcal{S}$, every gate with three links is solely adjacent to gates with two links.

Solution Mapping

Recall that every solution in $F_{\text{C/F}}(I)$ assigns values to all input links $X_1, \ldots, X_n$ of circuit $\mathcal{S}$; every solution in $F(\Phi(I))$ assigns values to all variables in $\mathcal{X}$. Now, let $\mathsf{a} \in F(\Phi(I))$ and denote

$$\mathsf{a}(x^\lambda) := (\mathsf{a}(x_1^\lambda), \ldots, \mathsf{a}(x_n^\lambda))$$

for $\lambda \in \{0,1\}$. Function $\Psi(I, \mathsf{a})$ returns $\mathsf{a}(x^\lambda)$ if $c_{\text{C/F}}(\mathsf{a}(x^\lambda), I) > c_{\text{C/F}}(\mathsf{a}(x^{\bar{\lambda}}), I)$ for some $\lambda \in \{0,1\}$ and $\mathsf{a}(x^0)$ otherwise. This terminates the description of the reduction. ○

6.3.6 Proving the Correctness and Tightness of the Reduction

In this subsection, we prove the correctness and tightness of our reduction via a sequence of lemmas. In Lemmas 6.2–6.6 and Corollary 6.1, we prove *properties of a locally optimal solution* $\mathsf{a} \in F(\Phi(I))$. Unless otherwise mentioned, let $\lambda \in \{0,1\}$, in the following.

Roadmap of the Proof In Lemma 6.2, we first focus on the set of predicates which are trivially satisfied in a and derive properties for the involved variables in Corollary 6.1. In Lemma 6.3, we show the connection between variables b, comp_i, and $\text{comp}^\star_j$ for all $i \in [n]$ and $j \in [2 : 2n]$. In Lemma 6.4, we show that for $\lambda = v(b)$, all link variables in circuit $S^{\bar{\lambda}}$ are set to d; by Corollary 6.1, all link variables in circuit S^λ are set to binary values. In Lemma 6.5, we prove that best $= e$. We close our proof by explicitly stating the variable assignment in a in Definition 6.1 and show in Lemma 6.6 that any deviating assignment cannot be locally optimal.

Lemma 6.2. *All predicates in* (6.1)*,* (6.3)*,* (6.4)*,* (6.11)*, and* (6.13)*–*(6.18) *are satisfied.*

Proof. Note that in each of the above predicates, at least one variable appears for the first time, with respect to the given order. In detail, $l_2(g)$, resp. $l_3(g)$ in (6.1) for all $g \in \tilde{\mathcal{G}}$ and $\text{comp}^\star_{2i-1}$ in $\mathsf{A}^0_{\text{carry}(Q_i)}$ in (6.3), and $\text{comp}^\star_{2i}$ in $\mathsf{A}^1_{\text{carry}(P_i)}$ in (6.4) for all $i \in [2:n]$. Furthermore, $x_i^{\kappa+2}$ in (6.13), $l_2(g)$, resp. $l_3(g)$ in (6.14), b in (6.15), c in (6.17), for all $i \in [n]$, $\kappa \in \{0,1\}$, and $g \in \mathcal{G}^2$. Thus, the value of the variable appearing for the first time can be set to satisfy the predicate and only predicates of lower weight may become violated. In predicates (6.11), (6.16), and (6.18) for propagation trees, the value of each child node can be set to the value of the parent node and only predicates of lower weight are violated. From this, the claim follows by induction. □

Corollary 6.1. *The following properties hold:*

1. *For all $i \in [n]$ and $\kappa \in \{0,1\}$, $x_i^\kappa = x_i^{\kappa+2}$.*
2. *For all $\ell \in \mathcal{X}_\mathcal{L}^\kappa$ and $\kappa \in \{0,1\}$, $v(\ell) \in \{d, \mathsf{R}(\ell,(x^\kappa))\}$; for all $\ell \in \mathcal{X}_\mathcal{L}^2$, $v(\ell) = \mathsf{R}(\ell,(x^0,x^1)\}$.*
3. *If $v(\ell) = d$ for some $\ell \in \tilde{\mathcal{X}}_\mathcal{L}$, then $v(\ell_s) = d$ for all $\ell_s \in \tilde{\mathcal{X}}_\mathcal{L}$ which are successors of ℓ in the topological order of $\tilde{\mathcal{G}}$.*
4. *There exists some $i \in [2:2n-2]$ such that $comp_j^\star = e$ for all $j \in [i-1]$ and $comp_j^\star = comp_{2n}^\star$ for all $j \in [i:2n]$.*
5. *For all $i \in [n]$, $\kappa \in \{0,1\}$, and $t \in V_{T_1}$, $v(t) = v(a_i^\kappa) = v(a) = v(a') = comp_{2n}^\star$; for all $(1,1)$-gates $g \in \tilde{\mathcal{G}}$ and $u \in V_{T_2}$, $v(u) = v(b_g) = v(b') = v(b)$; for all $i \in [0:n-1]$ and $w \in V_{T_3}$, $v(w) = v(c_i) = v(c)$.*
6. *If $\lambda = v(b)$, then $v(l) \in \{0,1\}$ for all $l \in \mathcal{X}_\mathcal{L}^\lambda$.*
7. *All predicates in* (6.2) *are satisfied.*
8. *Let $\lambda = v(b)$. For all $i \in [2:n]$, if $comp_{2i-1}^\star = \lambda$ then $comp_i = comp_{2i-1}^\star$.*
9. *Predicate $\mathsf{A}^3_{carry(P_1)}$ in* (6.7) *is satisfied; for all $i \in [2:n-1]$, if $comp_{2i-1}^\star = e$, then predicate $\mathsf{A}^3_{carry(P_i)}$ in* (6.7) *is satisfied.*

Proof. 1. Follows since all predicates $\mathsf{B}_{\mathrm{copy}(g_{x_i^\kappa})}$ in (6.13) for all $i \in [n]$ and $\kappa \in \{0,1\}$ are satisfied by Lemma 6.2.

2. For all $\tilde{g} \in \tilde{\mathcal{G}}$, $\check{g} \in \mathcal{G}^2$, and $\mu, \kappa \in \{0,1\}$, predicates $\mathsf{A}^\mu_{\mathrm{corr}(\tilde{g})}$, respectively $\mathsf{A}_{\mathrm{corr}(\tilde{g})}$ in (6.1), $\mathsf{B}_{\mathrm{copy}_{\tilde{g}_i^\kappa}}$ in (6.13) and $\mathsf{B}^\mu_{\mathrm{corr}(\check{g})}$, respectively $\mathsf{B}_{\mathrm{corr}(\check{g})}$ in (6.14) are satisfied because of Lemma 6.2. From this, the claim follows by induction.

3. For all $g \in \tilde{\mathcal{G}}$ and $\mu \in \{0,1\}$, predicates $\mathsf{A}^\mu_{\mathrm{corr}(g)}$, respectively $\mathsf{A}_{\mathrm{corr}(g)}$ in (6.1) are satisfied because of Lemma 6.2. This implies for every $g \in \tilde{\mathcal{G}}$, that if $l(g) = d$ for an input variable $l(g)$, then also $l_s(g) = d$ for all output variables $l_s(g)$ of g. From this, the claim follows by induction.

4. Assume that such an index i does not exist. This implies that there exists some $j \in [2 : 2n-1]$ such that $\text{comp}^\star_j \in \{0,1\}$ and $\text{comp}^\star_{j+1} \neq \text{comp}^\star_j$. If $j = 2r$ for some $r \in [n-1]$ then predicate $\mathsf{A}^0_{\text{carry}(Q_{r+1})}$ in (6.3) is not satisfied; if $j = 2r+1$ for some $r \in [n-1]$ then predicate $\mathsf{A}^1_{\text{carry}(P_{r+1})}$ in (6.4) is not satisfied. In both cases, a contradiction to Lemma 6.2.

5. All predicates for trees T_i with $i \in [3]$ in (6.11), (6.16), and (6.18) are satisfied by Lemma 6.2. From this, the claim follows by induction.

6. Suppose that the property does not hold. Let $g \in \mathcal{G}^\lambda$ be the smallest gate with respect to the topological sorting of $\mathcal{G}^\lambda$ which has an output variable $\ell \in \mathcal{X}^\lambda_{\mathcal{L}}$ with $v(\ell) = d$. Then $v(\ell') \in \{0,1\}$ for all input variables ℓ' of g. Three cases have to be considered:

   a) If $\ell = z^\lambda_i$ for some $i \in [n]$, then setting z^λ_i to its correct value in $\{0,1\}$ does not violate the corresponding predicate $\mathsf{A}_{\text{corr}(g)}$ in (6.1), $\mathsf{A}_{\text{corr}(D_i)}$ in (6.2), and $\mathsf{A}_{\text{out}(D_i)}$ in (6.6) and improves $\mathsf{C}^2_{\text{value}(g)}$ in (6.23) and therefore improves the solution

   b) If $l = y^\lambda_i$ for some $i \in [n]$, then setting y^λ_i to its correct value in $\{0,1\}$ does not violate the corresponding predicate $\mathsf{A}_{\text{corr}(g)}$ in (6.1), $\mathsf{B}^\lambda_{\text{load}_i}$ in (6.12) and improves $\mathsf{C}^2_{\text{value}(g)}$ in (6.23), thus improving the solution.

   c) Otherwise, ℓ is an input variable of some gate g'. Note that if g is not a $(1,1)$-gate, then g' is a $(1,1)$-gate. By definition, $v(\ell_s) = d$ for all output variables ℓ_s of gate g'. Setting ℓ to its correct value in $\{0,1\}$ does not violate the corresponding predicate $\mathsf{A}^\mu_{\text{corr}(g)}$, respectively $\mathsf{A}_{\text{corr}(g)}$ and $\mathsf{A}^\mu_{\text{corr}(g')}$, respectively $\mathsf{A}_{\text{corr}(g')}$ in (6.1) for $\mu \in \{0,1\}$ and improves $\mathsf{C}^2_{\text{value}(g)}$ in (6.23) if ℓ is the output variable of a $(1,1)$-gate or improves $\mathsf{C}^1_{\text{value}(g')}$ in (6.23) otherwise, thus improving the solution.

7. By (6), $z^\lambda_i \in \{0,1\}$ for all $i \in [n]$, where $\lambda = v(b)$. Furthermore, for all $i \in [n]$, variable comp_i occurs in predicate $\mathsf{A}_{\text{corr}(D_i)}$ in (6.2) for the first time, with respect to the given order. Hence, comp_i can be set to satisfy predicate $\mathsf{A}_{\text{corr}(D_i)}$ in (6.2) and only predicates of lower weight are violated.

8. By (6), $z^\lambda_j \in \{0,1\}$ for all $j \in [n]$, where $\lambda = v(b)$. Assume there exists some $i \in [2 : n]$ such that $\text{comp}^\star_{2i-1} = \lambda$ and $\text{comp}_i \neq \lambda$. By (4), $\text{comp}^\star_{2i} = \text{comp}^\star_{2i-1} = \lambda$. If $\text{comp}_i = \bar{\lambda}$, then setting $\text{comp}_i \leftarrow \lambda$ does not violate heavier predicate $\mathsf{A}_{\text{corr}(D_i)}$ in (6.2), improves predicate $\mathsf{A}^2_{\text{carry}(P_i)}$ in (6.5), and only predicates of lower weight are violated. If $\text{comp}_i = e$, then setting $\text{comp}_i \leftarrow \lambda$ does not violate heavier predicates $\mathsf{A}_{\text{corr}(D_i)}$ in (6.2), $\mathsf{A}^2_{\text{carry}(P_i)}$ in (6.5), $\mathsf{A}_{\text{out}(D_i)}$ in (6.6), $\mathsf{A}^3_{\text{carry}(P_i)}$ in (6.7), $\mathsf{A}^1_{\text{care}(P_i)}$ in (6.10) and improves predicate $\mathsf{C}_{\text{incentive}(P_i)}$ in (6.22).

9. First, consider the case $i = 1$. If $\mathsf{A}^3_{\text{carry}(P_1)}$ in (6.7) is unsatisfied then this implies that $\text{comp}_1 \in \{0,1\}$ and $\text{comp}^\star_2 \neq \text{comp}_1$. Setting $\text{comp}^\star_2 \leftarrow \text{comp}_1$ improves

predicate $\mathsf{A}^3_{\text{carry}(P_1)}$ in (6.7) and only violates predicates of smaller weight. Now, assume that predicate $\mathsf{A}^3_{\text{carry}(P_i)}$ in (6.7) is unsatisfied for some $i \in [2 : n-1]$ with $\text{comp}^\star_{2i-1} = e$. This implies that $\text{comp}_i \in \{0,1\}$, $\text{comp}^\star_{2i} \neq \text{comp}_i$, and predicate $\mathsf{A}^1_{\text{carry}(P_i)}$ in (6.4) is satisfied. Setting $\text{comp}^\star_{2i} \leftarrow \text{comp}_i$ improves $\mathsf{A}^3_{\text{carry}(P_i)}$ in (6.7), does not violate heavier predicate $\mathsf{A}^1_{\text{carry}(P_i)}$ in (6.4), and only violates predicates of smaller weight.

□

Lemma 6.3. *$comp_i = comp^\star_j = v(b)$ for all $i \in [n], j \in [2 : 2n]$.*

Proof. By Corollary 6.1 (1), $x_i^\kappa = x_i^{\kappa+2}$ for all $i \in [n]$ and $\kappa \in \{0,1\}$. By Corollary 6.1 (2), the value computed in variable best is correct with respect to the input. Recall that by definition of S^2, if best $= \lambda \in \{0,1\}$ then there exists some pos $\in [n]$, such that $Z(x^\lambda)_{\text{pos}} > Z(x^{\bar\lambda})_{\text{pos}}$ and $Z(x^\lambda)_r = Z(x^{\bar\lambda})_r$ for all $r \in [\text{pos}-1]$; if best $= e$ then for all $r \in [n]$, $Z(x^\lambda)_r = Z(x^{\bar\lambda})_r$ and for simplicity reasons, we will treat this as pos $= n+1$.

By Corollary 6.1 (4) there exists some $p \in [2 : 2n-2]$ such that $\text{comp}^\star_j = e$ for all $j \in [p-1]$ and $\text{comp}^\star_j = \text{comp}^\star_{2n}$ for all $j \in [p : 2n]$. Corollary 6.1 (9) implies that if p is even, $p = 2q$, and $\text{comp}_q = \alpha \in \{0,1\}$ then $\text{comp}^\star_p = \alpha$. All predicates in (6.2) are satisfied by Corollary 6.1 (7) and therefore $\text{comp}_{\text{pos}} \in \{\lambda, \bar\lambda\}$ (only defined if pos $\leq n$) and hence, $p \leq 2\text{pos}$.

Now, let $\lambda = v(b)$. By Corollary 6.1 (6), $z_i^\lambda \in \{0,1\}$ for all $i \in [n]$. Set $\mu = \text{comp}^\star_{2n}$. By Corollary 6.1 (5), $v(a) = v(a') = \mu$ and $v(b') = v(b) = \lambda$; furthermore predicate $\mathsf{B}_{\text{propagate}}$ in (6.15) is satisfied by Lemma 6.2. Predicate $\mathsf{B}_{\text{setIncentives}}$ in (6.17) is satisfied by Lemma 6.2 and therefore $v(c) = \lambda$ if $\mu = \lambda$ and $v(c) = e$ otherwise. By Corollary 6.1 (5), $v(c_j) = v(c)$ for all $j \in [0 : n-1]$. In the following, we consider five cases. In the first case, we show that $\text{comp}_i = \text{comp}^\star_j = v(b)$ holds for all $i \in [n]$ and $j \in [2 : 2n]$. Thus, the claim of the Lemma holds in this case. In all other cases, the solution can be improved.

1. Let $p = 2$, $\mu = \lambda$, and $\text{comp}_1 = \lambda$. By assumption, $\text{comp}^\star_{2t-1} = \lambda$ for all $t \in [2 : n]$. Then, Corollary 6.1 (8), implies that $\text{comp}_t = \text{comp}^\star_{2t-1}$.

2. Let $p = 2$, $\mu = \lambda$, and $\text{comp}_1 \neq \lambda$. Predicate $\mathsf{A}^3_{\text{carry}(P_1)}$ in (6.7) is satisfied by Corollary 6.1 (9) and therefore $\text{comp}_1 = e$. Setting $\text{comp}_1 \leftarrow \lambda$ does not violate heavier predicates $\mathsf{A}_{\text{corr}(D_1)}$ in (6.2), $\mathsf{A}_{\text{out}(D_1)}$ in (6.6), $\mathsf{A}^3_{\text{carry}(P_1)}$ in (6.7), and $\mathsf{A}^1_{\text{care}(P_1)}$ in (6.10), improves $\mathsf{C}_{\text{incentive}(P_1)}$ in (6.21) and therefore improves the solution.

3. Let $p \in [3 : 2\text{pos}]$ and $\mu = \lambda$. By definition, $\text{comp}^\star_{p-1} = e$. The proof now splits on the parity of $p-1$:

   a) $p-1$ is even, $p-1 = 2q$. Predicate $\mathsf{A}^3_{\text{carry}(P_q)}$ in (6.7) is satisfied by Corollary 6.1 (9) and therefore $\text{comp}_q = e$. Setting $\text{comp}^\star_{p-1} \leftarrow \lambda$ does not violate heavier predicates $\mathsf{A}^1_{\text{carry}(P_q)}$ in (6.4), $\mathsf{A}^3_{\text{carry}(P_q)}$ in (6.7), $\mathsf{A}^0_{\text{carry}(Q_{q+1})}$

in (6.3), $\mathsf{A}^0_{\text{care}(P_q)}$ in (6.8), $\mathsf{A}_{\text{care}(Q_{q+1})}$ in (6.9), $\mathsf{A}^1_{\text{care}(P_q)}$ in (6.10), improves $\mathsf{C}^0_{\text{incentive}(Q_{q+1})}$ in (6.19) and thus improves the solution.

b) $p-1$ is odd, $p = 2q$. Predicate $\mathsf{A}^3_{\text{carry}(P_q)}$ in (6.7) is satisfied by Corollary 6.1 (9) and therefore $\text{comp}_q \in \{\lambda, e\}$. Recall that comp_n was defined as a binary variable, thus $\text{comp}_q = \lambda$ if $q = n$. Setting $\text{comp}^\star_{p-1} \leftarrow \lambda$ does not violate heavier predicates $\mathsf{A}^0_{\text{carry}(Q_q)}$ in (6.3), $\mathsf{A}^1_{\text{carry}(P_q)}$ in (6.4), $\mathsf{A}^2_{\text{carry}(P_q)}$ in (6.5), $\mathsf{A}^0_{\text{care}(P_q)}$ in (6.8), $\mathsf{A}_{\text{care}(Q_q)}$ in (6.9), improves predicate $\mathsf{C}^1_{\text{incentive}(Q_q)}$ in (6.20) and may only violate the smaller predicate $\mathsf{C}_{\text{incentive}(P_q)}$ in (6.22) and thus improves the solution.

4. Let $p \in [2 : 2\text{pos} - 1]$ and $\mu = \bar{\lambda}$. Predicate $\mathsf{B}_{\text{propagate}}$ in (6.15) is satisfied by Lemma 6.2 and therefore best $\neq e$; thus, $\text{pos} < n + 1$. Recall that by definition, $\text{comp}^\star_{p-1} = e$ and $\text{comp}^\star_p = \mu$. The proof now splits on the parity of p:

   a) p is odd, $p+1 = 2q$. Setting $\text{comp}^\star_p \leftarrow e$ does not violate heavier predicates $\mathsf{A}^0_{\text{carry}(Q_q)}$ in (6.3), $\mathsf{A}^1_{\text{carry}(P_q)}$ in (6.4), $\mathsf{A}^2_{\text{carry}(P_q)}$ in (6.5), $\mathsf{A}^0_{\text{care}(P_q)}$ in (6.8), $\mathsf{A}_{\text{care}(Q_q)}$ in (6.9), improves $\mathsf{C}^1_{\text{incentive}(Q_q)}$ in (6.20), may only violate the smaller predicate $\mathsf{C}_{\text{incentive}(P_q)}$ in (6.22), and thus improves the solution.

   b) p is even, $p = 2q$. Predicate $\mathsf{A}^3_{\text{carry}(P_q)}$ in (6.7) is satisfied by Corollary 6.1 (9) and this implies that $\text{comp}_q \in \{e, \mu\}$. The proof now splits on comp_q.

      i. If $\text{comp}_q = e$, then setting $\text{comp}^\star_p \leftarrow e$ does not violate heavier predicates $\mathsf{A}^1_{\text{carry}(P_q)}$ in (6.4) (only defined if $q > 1$), $\mathsf{A}^3_{\text{carry}(P_q)}$ in (6.7), $\mathsf{A}^0_{\text{carry}(Q_{q+1})}$ in (6.3), $\mathsf{A}^0_{\text{care}(P_q)}$ in (6.8), $\mathsf{A}_{\text{care}(Q_{q+1})}$ in (6.9), $\mathsf{A}^1_{\text{care}(P_q)}$ in (6.10), improves $\mathsf{C}^0_{\text{incentive}(Q_{q+1})}$ in (6.19) and therefore improves the solution.

      ii. Now, let $\text{comp}_q = \mu$. By definition, $Z(x^0)_q = Z(x^1)_q$. Corollary 6.1 (2) (6) (7) imply that $z^\lambda_q = z^{\bar{\lambda}}_q \in \{0, 1\}$. Setting $\text{comp}_q \leftarrow e$ does not violate heavier predicates $\mathsf{A}_{\text{corr}(D_q)}$ in (6.2), $\mathsf{A}^2_{\text{carry}(P_q)}$ in (6.5) (only defined if $q > 1$), $\mathsf{A}_{\text{out}(D_q)}$ in (6.6), $\mathsf{A}^3_{\text{carry}(P_q)}$ in (6.7), and $\mathsf{A}^1_{\text{care}(P_q)}$ in (6.10). If $q > 1$, predicate $\mathsf{C}_{\text{incentive}(P_q)}$ in (6.22) improves, otherwise predicate $\mathsf{C}_{\text{incentive}(P_1)}$ in (6.21) improves and thus improves the solution.

5. Let $p = 2\text{pos}$ and $\mu = \bar{\lambda}$. Recall that by definition of circuit $\mathcal{S}$, if $\text{pos} < n + 1$, then $\text{pos} \in [2 : n - 1]$. Predicate $\mathsf{A}^3_{\text{carry}(P_{\text{pos}})}$ in (6.7) is satisfied by Corollary 6.1 (9), and $\mathsf{A}_{\text{corr}(D_{\text{pos}})}$ in (6.2) is satisfied by Corollary 6.1 (7), therefore $\text{comp}_{\text{pos}} = \bar{\lambda}$. By definition, $\text{comp}^\star_{p-1} = e$. By Corollary 6.1 (2) (6) (7), $z^\lambda_{\text{pos}} = 1$ and $z^{\bar{\lambda}}_{\text{pos}} = 0$. Setting $\text{comp}_{\text{pos}} \leftarrow \lambda$ does not violate the heavier predicates $\mathsf{A}_{\text{corr}(D_{\text{pos}})}$ in (6.2) and $\mathsf{A}^2_{\text{carry}(P_{\text{pos}})}$ in (6.5), improves predicate $\mathsf{A}_{\text{out}(D_{\text{pos}})}$ in (6.6) and only predicates of lower weight become violated.

□

Lemma 6.4. *Set $\lambda = v(b)$. Then $v(l) = d$ for all $l \in \mathcal{X}_{\mathcal{L}}^{\bar{\lambda}} \setminus \{x_i^{\bar{\lambda}} \mid i \in [n]\}$, and $x_i^{\bar{\lambda}} = y_i^{\lambda}$ for all $i \in [n]$.*

Proof. By Lemma 6.3, $\text{comp}_{2n}^{\star} = v(b) = \lambda$. By Corollary 6.1 (5), $v(a_i^{\kappa}) = \text{comp}_{2n}^{\star} = \lambda$ for all $i \in [n]$, $\kappa \in \{0,1\}$, and $v(b') = v(b_g) = v(b) = \lambda$ for all $(1,1)$-gates $g \in \tilde{\mathcal{G}}$. Consider the two claims of the lemma:

1. Let $g \in \mathcal{G}^{\bar{\lambda}}$ be the largest gate with respect to the topological sorting of $\mathcal{G}^{\bar{\lambda}}$ which has an output variable ℓ with $v(\ell) \in \{0,1\}$. Three cases have to be considered:
   a) If $\ell = z_i^{\bar{\lambda}}$ for some $i \in [n]$ then, since $z_i^{\lambda} \in \{0,1\}$ by Corollary 6.1 (6) and $\text{comp}_i = \lambda$ for all $i \in [n]$ by Lemma 6.3, setting $z_i^{\bar{\lambda}} \leftarrow d$ does not violate the corresponding predicate $\mathsf{A}_{\text{corr}(g)}$ in (6.1), $\mathsf{A}_{\text{corr}(D_i)}$ in (6.2), and $\mathsf{A}_{\text{out}(D_i)}$ in (6.6), improves $\mathsf{C}^2_{\text{value}(g)}$ in (6.23) and therefore improves the solution.
   b) If $\ell = y_i^{\bar{\lambda}}$ for some $i \in [n]$ then setting $y_i^{\bar{\lambda}} \leftarrow d$ does not violate heavier predicates $\mathsf{A}_{\text{corr}(g)}$ in (6.1) and $\mathsf{B}^{\bar{\lambda}}_{\text{load}_i}$ in (6.12) since $v(a_i^{\lambda}) = \lambda$, improves predicate $\mathsf{C}^2_{\text{value}(g)}$ in (6.23) and therefore improves the solution.
   c) Otherwise, ℓ is an input variable of some gate g'. Note that if g is not a $(1,1)$-gate, then g' is a $(1,1)$-gate. By definition, $v(\ell_s) = d$ for all output variables ℓ_s of gate g'. Setting $v(\ell) \leftarrow d$ does not violate heavier predicates $\mathsf{A}^{\mu}_{\text{corr}(g)}$, respectively $\mathsf{A}_{\text{corr}(g)}$ and $\mathsf{A}^{\mu}_{\text{corr}(g')}$, respectively $\mathsf{A}_{\text{corr}(g')}$ in (6.1) for $\mu \in \{0,1\}$ and improves $\mathsf{C}^2_{\text{value}(g)}$ in (6.23), if ℓ is the output variable of a $(1,1)$-gate or improves $\mathsf{C}^1_{\text{value}(g')}$ in (6.23) otherwise, thus improving the solution.
2. Assume there exists some $i \in [n]$ such that $x_i^{\bar{\lambda}} \neq y_i^{\lambda}$. As shown above, $v(a_i^{\bar{\lambda}}) = \lambda$. Thus, predicate $\mathsf{B}^{\lambda}_{\text{load}_i}$ in (6.12) is unsatisfied. Setting $x_i^{\bar{\lambda}} \leftarrow y_i^{\lambda}$ improves predicate $\mathsf{B}^{\lambda}_{\text{load}_i}$ in (6.12) and does not violate heavier predicates (6.1), since $v(l) = d$ for all $l \in \mathcal{X}_{\mathcal{L}}^{\bar{\lambda}} \setminus \{x_i^{\bar{\lambda}} \mid i \in [n]\}$.

□

Lemma 6.5. *best $= e$.*

Proof. Assume that best $= \lambda \in \{0,1\}$. Then, $c_{\mathrm{C/F}}(x^{\lambda}, I) > c_{\mathrm{C/F}}(x^{\bar{\lambda}}, I)$ by Corollary 6.1, (2) and the definition of circuit S^2. By Lemma 6.4, $x_i^{\bar{\lambda}} = y_i^{\lambda}$ for all $i \in [n]$. But $c_{\mathrm{C/F}}((y_i^{\lambda})_{i=1}^{n}, I) \geq c_{\mathrm{C/F}}(x^{\lambda}, I)$ by definition of variables y_i. A contradiction. □

Definition 6.1. *We define a* final assignment $B \in F_{(3,2,3)\text{-MCA}_{3\text{-}par}}(I)$ *to fulfill the following conditions: Set $\lambda = v(b)$. For all $i \in [n], j \in [2:2n]$, $comp_i = comp_j^{\star} = \lambda$, $best = e$, $x_i^{\bar{\lambda}} = y_i^{\lambda}$, $x_i^0 = x_i^1 = x_i^2 = x_i^3$. For all $k \in \{a, a', b', c\}$, $w \in V_{T_1} \cup V_{T_2} \cup V_{T_3}$, $i \in [n]$, $j \in [0:n-1]$, $\kappa \in \{0,1\}$, and $(1,1)$-gates $g \in \tilde{\mathcal{G}}$, $v(w) = v(k) = v(a_i^{\kappa}) =$*

$v(b_g) = v(c_j) = \lambda$. For all $l \in \mathcal{X}_{\mathcal{L}}^{\lambda}$, $v(l) = \mathsf{R}(l, (x^{\lambda}))$. For all $l \in \mathcal{X}_{\mathcal{L}}^{\bar{\lambda}} \setminus \{x_i^{\bar{\lambda}} \mid i \in [n]\}$, $v(l) = d$. For all $l \in \mathcal{X}_{\mathcal{L}}^{2}$, $v(l) = \mathsf{R}(l, (x^0, x^1))$.

Lemma 6.6. *Every locally optimal assignment is a final assignment.*

Proof. Assume that some assignment A is locally optimal with $v(b) = \lambda$. By Lemma 6.5, best $= e$. By Lemma 6.3, $\text{comp}_i = \text{comp}_j^{\star} = \lambda$ for all $i \in [n], j \in [2 : 2n]$. By Lemma 6.4, $x_i^{\bar{\lambda}} = y_i^{\lambda}$ for all $i \in [n]$. For all $l \in \mathcal{X}_{\mathcal{L}}^{\lambda}$, $v(l) = \mathsf{R}(l, (x^{\lambda}))$ by Corollary 6.1 (2), (6). For all $l \in \mathcal{X}_{\mathcal{L}}^{\bar{\lambda}} \setminus \{x_i^{\bar{\lambda}} \mid i \in [n]\}$, $v(l) = d$ by Lemma 6.4. For all $l \in \mathcal{X}_{\mathcal{L}}^{2}$, $v(l) = \mathsf{R}(l, (x^0, x^1))$ by Corollary 6.1 (2). By Corollary 6.1 (5), $v(t) = v(a) = v(a') = v(a_i^{\kappa}) = \text{comp}_{2n}^{\star} = \lambda$ for all $i \in [n]$, $\kappa \in \{0,1\}$, $t \in V_{T_1}$, and $v(u) = v(b') = v(b_g) = \lambda$ for all $(1,1)$-gates $g \in \tilde{\mathcal{G}}$ and $u \in V_{T_2}$. Predicate $\mathsf{B}_{\text{setIncentives}}$ in (6.17) is satisfied by Lemma 6.2 and this implies that $v(c) = \lambda$. By Corollary 6.1 (5), $v(w) = v(c_i) = \lambda$ for all $i \in [0 : n-1]$ and $w \in V_{T_3}$. Variable best $= e$ and for all $i \in [n]$, $x_i^{\bar{\lambda}} = y_i^{\lambda}$ by Lemma 6.4, therefore $x_i^0 = x_i^1$ by definition of variables y_i^{κ} for all $\kappa \in \{0,1\}$. By Corollary 6.1 (1), $x_i^{\kappa} = x_i^{\kappa+2}$ for all $i \in [n], \kappa \in \{0,1\}$ and this implies that $x_i^0 = x_i^1 = x_i^2 = x_i^3$. □

Why Proving Tightness is Non-Trivial

Let I be an instance of Circuit/Flip and let $\mathsf{x} \in \{0,1\}^n$ be a feasible solution of I such that $\text{Improve}(\text{Improve}(\mathsf{x}, I), I)$ is not a locally optimal solution of I. Let x and $\mathsf{x}' := \text{Improve}(\mathsf{x}, I)$ differ in bit k_1 and let x' and $\mathsf{x}'' := \text{Improve}(\mathsf{x}', I)$ differ in bit k_2 with $k_1 \neq k_2$. Our construction $\Phi(I)$ allows (with appropriate settings of the respective variables) a sequence of improving steps which leads from an assignment $\mathsf{a} \in F(\Phi(I)$ with $\mathsf{a}(x^{\lambda}) := \mathsf{x}$ and $\mathsf{a}(x^{\bar{\lambda}}) := \mathsf{x}'$ to an assignment $\mathsf{a}' \in F(\Phi(I)$ with $\mathsf{a}'(x^{\lambda}) := \mathsf{x}''$ and $\mathsf{a}'(x^{\bar{\lambda}}) := \mathsf{x}'$. Note that solutions x and x'' differ in two bits. One of these bits has to be changed first in x^{λ} and this might be bit k_2. Let $\mathsf{x}''' \in \{0,1\}^n$ be the solution that differs from x in bit k_2. We have no information about $\text{Cost}_{\text{C/F}}(\mathsf{x}''', I)$. We cannot exclude the case that $\text{Cost}_{\text{C/F}}(\mathsf{x}''', I) < \text{Cost}_{\text{C/F}}(\mathsf{x}, I)$ and that $\text{Improve}(\mathsf{x}''', I)$ is a locally optimal solution of I. We design the set $\mathscr{R} \subseteq F(\Phi(I)$ in such a way that starting from an assignment $\mathsf{a} \in \mathscr{R}$ with $\mathsf{a}(x^{\bar{\lambda}}) = \text{Improve}(\mathsf{a}(x^{\lambda}), I)$, the assignment of $x^{\bar{\lambda}}$ can only be changed after the assignment of x^{λ} has been set to $\text{Improve}(\mathsf{a}(x^{\bar{\lambda}}), I)$.

Our proof of tightness relies in large parts on predicates (6.8)–(6.10) and the following assumptions on circuit $\mathcal{S}$ in the given instance I of Circuit/Flip, we made in Section 6.2: Recall that we assumed that $c_{\text{C/F}}(\mathsf{x}, I) \neq c_{\text{C/F}}(\mathsf{x}', I)$ for solutions $\mathsf{x}, \mathsf{x}' \in F_{\text{C/F}}(I)$ with $\mathsf{x} \neq \mathsf{x}'$. Furthermore, for every input link X_i with $i \in [n]$ and every output link Z_j with $j \in [n]$, there exists a path in circuit $\mathcal{S}$ from X_i to Z_j. For output links Z_1 and Z_n, we assumed that $Z(\mathsf{x})_1 = Z(\mathsf{x})_n = 0$ for all $\mathsf{x} \in F_{\text{C/F}}(I)$.

Definition 6.2 (The Set $\mathscr{R}$)**.** *For a solution $\mathsf{a} \in F(\Phi(I))$, denote $\mathsf{a}(comp^{\star}) := (\mathsf{a}(comp_i^{\star}))_{i=2}^{2n-1}$. For $k \in [n]$, denote $succ(k) := \{l \in \mathcal{L} \mid l$ is on a path from input link X_k to some output link $Z_i, i \in [n]\}$ and let $\mathcal{X}_{succ(k)}^{\lambda}$ denote the set of link variables* *in $\mathcal{X}^\lambda_\mathcal{L}$ defined by links from succ(k). We define the following predicates, where for $k \in [n]$, $\lambda \in \{0,1\}$*

$$Q_1(\mathsf{a},\lambda,k) := [\mathsf{a}(l) = d \ \forall\ l \in \mathcal{X}^\lambda_{succ(k)}]$$
$$Q_2(\mathsf{a},\lambda) := [\mathsf{a}(comp^\star) = \lambda^{2n-2} \wedge \mathsf{a}(comp_i) = \lambda\ \forall i \in [n]].$$

We define $\mathscr{R}$ to be the set of all solutions $\mathsf{a} \in F(\Phi(I))$ which satisfy the following properties:

1. *All predicates in* (6.1) *and* (6.2) *are satisfied.*
2. *There exists some $\lambda \in \{0,1\}$ such that*
   a) $\mathsf{a}(x^{\bar\lambda}) = \textsc{Improve}(\mathsf{a}(x^\lambda), I)$,
   b) *$Q_1(\mathsf{a},\bar\lambda,k)$ is satisfied for some $k \in [n]$, and*
   c) *$Q_2(\mathsf{a},\lambda)$ is satisfied.*

In the following Lemma, we show that for every sequence of improving solution which starts in some solution in $\mathscr{R}$, the vector comp* is, loosely speaking, of the form $e^*\bar\lambda^*\lambda^*$. For this, we solely focus on the set of predicates (6.3)–(6.10) and prove the claim by induction on the number of improving steps.

Lemma 6.7. *Let $\lambda \in \{0,1\}$ and let σ be a sequence of improving solutions from $F(\Phi(I))$, where for the first solution a_0 in σ items* (1), (2b), *and* (2c) *from Definition 6.2 are satisfied and for each solution $\mathsf{a} \in \sigma$, $\mathsf{a}_0(x^\kappa) = \mathsf{a}(x^\kappa)$ for all $\kappa \in \{0,1\}$. Fix some $\mathsf{a} \in \sigma$ and refer to pos(a_0) by pos. Then, all predicates in* (6.1) *and* (6.2) *are satisfied in a and*

$$\mathsf{a}(comp^\star) = uvw, \text{ where } u \in e^*, v \in \bar\lambda^*, w \in \lambda^* \tag{INV}$$

with

1. $|u| \le 2 \cdot pos - 2$ *and if* $|v| > 0$ *then* $|u| + |v| \ge 2 \cdot pos - 1$
2. *For all $i \in [n-1]$ with $i < pos$ or $comp^\star_{2i-1} = \lambda$, $\mathsf{A}^2_{carry(P_i)}$ in* (6.5) *is satisfied and $\mathsf{A}^3_{carry(P_i)}$ in* (6.7) *is satisfied if $i < n$*
3. *$\mathsf{A}^2_{carry(P_{pos})}$ in* (6.5) *is satisfied and if $\mathsf{a}(comp^\star_{2pos}) = \bar\lambda$ then $\mathsf{a}(comp_{pos}) = \bar\lambda$*

Proof. We prove the lemma by induction on the number of improving steps. By definition, (INV) is satisfied for a_0. Now, let a be some solution in σ. Since a is fixed in the following, we omit the assignment, for sake of readability, where it is clear from the context. We show that every step which violates (INV) does not improve a. For this, we fix some $i \in [n]$ and investigate all changes to comp$_i$, comp$^\star_{2i-1}$, and comp$^\star_{2i}$. These variables occur in predicates $\mathsf{A}_{\text{corr}(D_i)}$ in (6.2), $\mathsf{A}^0_{\text{carry}(Q_i)}$ in (6.3), $\mathsf{A}^1_{\text{carry}(P_i)}$ in (6.4), $\mathsf{A}^2_{\text{carry}(P_i)}$ in (6.5), $\mathsf{A}_{\text{out}(D_i)}$ in (6.6), $\mathsf{A}^3_{\text{carry}(P_i)}$ in (6.7), $\mathsf{A}^0_{\text{care}(P_i)}$ in (6.8), $\mathsf{A}_{\text{care}(Q_i)}$ in (6.9), and $\mathsf{A}^1_{\text{care}(P_i)}$ in (6.10) and in predicates of lower weight.

For sake of readability, we refer to these predicates only by their number. A predicate with the number (X) but defined by the parameter $i-1$ is denoted by $(X)^-$.

By assumption, all predicates in (6.1) and (6.2) are satisfied in a_0 and as they are the heaviest among the list of predicates, they remain satisfied throughout σ. Hence, we skip predicates $\mathsf{A}_{\mathrm{corr}(D_i)}$ in (6.2) for all $i \in [n]$, when listing satisfied predicates. Furthermore, all predicates in (6.2) being satisfied implies that if for some $i \in [n]$, $\mathrm{comp}_i = e$ then $z_i^0 = z_i^1 \in \{0,1\}$ and if $\mathrm{comp}_i = \mu \in \{0,1\}$ then $z_i^\mu \in \{0,1\}$.

First, consider changes of comp$_i$, $i \in [n]$. By (INV), predicate (6.5) is satisfied and as it is the largest predicate after predicate (6.2) in which comp_i appears, it remains satisfied in each improving step. Predicate (6.7) can only become violated by improving predicate (6.6). If $\mathrm{comp}^\star_{2i-1} = \mu \in \{0,1\}$, then predicate (6.5) being satisfied implies that $\mathrm{comp}_i = \beta \in \{e, \mu\}$. Changing the value of comp_i cannot improve predicate (6.6), since predicate (6.2) is satisfied. If $\mathrm{comp}^\star_{2i-1} = e$, then (INV) implies that $i \leq \mathrm{pos}$. We distinguish the cases $i < \mathrm{pos}$ and $i = \mathrm{pos}$. First, let $i < \mathrm{pos}$. Since $Z(x^0)_i = Z(x^1)_i$ and predicate (6.2) is satisfied, predicate (6.6) cannot improve by setting comp_i to a new value. Now, let $i = \mathrm{pos}$. As shown above, predicate (6.5) is satisfied after each improving step. By (INV), it is sufficient to consider the case $\mathrm{comp}_{2\mathrm{pos}} = \bar{\lambda}$. In this case, $\mathrm{comp}_{\mathrm{pos}} = \bar{\lambda}$ by (INV) and predicate (6.7) is satisfied. Now, predicates (6.5), (6.6), and (6.7) are satisfied and remain satisfied in each improving step.

Now, consider changes of variables comp$^\star_j$, $j = 2i-1$ or $j = 2i$. For sake of readability, we use the following notation: If $j = 2i-1$, then predicate (6.3/6.4) denotes predicate (6.3), predicate $(6.3/6.4)^-$ denotes predicate (6.4), and predicate (6.5/6.7) denotes predicate (6.5). If $j = 2i$, then predicate (6.3/6.4) denotes predicate (6.4), predicate $(6.3/6.4)^-$ denotes predicate $(6.3)^-$, and predicate (6.5/6.7) denotes predicate (6.7). For sake of presentation, we introduce an artificial variable $\mathrm{comp}^\star_1$, define $\mathrm{comp}^\star_1 = e$ to be constant, and naturally extend predicate (6.3/6.4) to $\mathrm{comp}^\star_1$. Note that this way predicate (6.3/6.4) is satisfied, regardless of the value of $\mathrm{comp}^\star_2$. Recall that by definition, $\mathrm{comp}^\star_{2n-2}, \mathrm{comp}^\star_{2n-1} \in \{0,1\}$ and ($\mathrm{pos} \in [2:n-1]$ or $\mathrm{pos} = n+1$). We distinguish the following eight cases, where $j \in [2:2n-1]$:

1. Let $\mathrm{comp}^\star_{j-1} = \mathrm{comp}^\star_j = e$ and $\mathrm{comp}^\star_{j+1} = \mu \in \{e,0,1\}$. (INV) implies that $i < \mathrm{pos}$ and predicates (6.3/6.4), $(6.3/6.4)^-$, and (6.5/6.7) are satisfied. If $\mu = e$, then setting $\mathrm{comp}^\star_j \leftarrow \beta \in \{0,1\}$ does not improve the solution, since predicate (6.3/6.4) remains satisfied and predicate $(6.3/6.4)^-$ becomes violated. Now, let $\mu \in \{0,1\}$. Predicate (6.5/6.7) is satisfied and therefore $\mathrm{comp}_j \in \{e, \mu\}$. Setting $\mathrm{comp}^\star_j \leftarrow \mu$ does not violate (INV), since predicates (6.3/6.4), $(6.3/6.4)^-$, and (6.5/6.7) remain satisfied. Setting $\mathrm{comp}^\star_j \leftarrow \bar{\mu}$ does not improve the solution, since predicate $(6.3/6.4)^-$ becomes violated.

2. Let $\mathrm{comp}^\star_{j-1} = e$, $\mathrm{comp}^\star_j = \mathrm{comp}^\star_{j+1} = \mu \in \{0,1\}$, and ($i < \mathrm{pos}$ or $i = \mathrm{pos}$ and $j = 2i-1$). Then, $j \leq 2n-2$. (INV) implies that predicates (6.3/6.4), $(6.3/6.4)^-$, and (6.5/6.7) are satisfied. Setting $\mathrm{comp}^\star_j \leftarrow \bar{\mu}$ does not improve the solution, since predicate $(6.3/6.4)^-$ becomes violated. Now, consider setting $\mathrm{comp}^\star_j \leftarrow e$.

First, let $j = 2i - 1$. Setting $\text{comp}_j^\star \leftarrow e$ does not violate (INV) since predicates (6.3/6.4), $(6.3/6.4)^-$, and (6.5/6.7) remain satisfied. Now, let $j = 2i$. Predicate (6.5/6.7) is satisfied and therefore $\text{comp}_i \in \{e, \mu\}$. If $\text{comp}_i = e$, then setting $\text{comp}_j^\star \leftarrow e$ does not violate (INV) since predicates (6.3/6.4), $(6.3/6.4)^-$, and (6.5/6.7) remain satisfied. If $\text{comp}_i = \mu$, then setting $\text{comp}_j^\star \leftarrow e$ does not improve the solution, since predicate (6.5/6.7) becomes violated.

3. Let $\text{comp}_{j-1}^\star = e$, $\text{comp}_j^\star = \bar{\lambda}$, $\text{comp}_{j+1}^\star = \mu \in \{0, 1\}$, and $j = 2\text{pos}$. Then, $j \leq 2n - 2$. By (INV), predicate (6.5) is satisfied, $\text{comp}_{\text{pos}} = \bar{\lambda}$ and thus, predicate (6.7) is satisfied. Setting $\text{comp}_j^\star \leftarrow \beta \in \{e, \lambda\}$ does not improve the solution, since predicate (6.7) becomes violated.

4. Let $\text{comp}_{j-1}^\star = e$, $\text{comp}_j^\star = \lambda$, and $j = 2\text{pos}$. Then, $j \leq 2n - 2$. (INV) implies that $\text{comp}_{j+1}^\star = \lambda$. Predicates (6.3/6.4)and $(6.3/6.4)^-$ are satisfied. Predicate (6.2) is satisfied and therefore $\text{comp}_{\text{pos}} \in \{0, 1\}$. First, let $\text{comp}_{\text{pos}} = \lambda$. This implies that predicate (6.7) is satisfied. Setting $\text{comp}_j^\star \leftarrow \mu \in \{e, \bar{\lambda}\}$ does not improve the solution, predicate (6.7) becomes violated. Now, let $\text{comp}_{\text{pos}} = \bar{\lambda}$. This implies that predicate (6.7) is unsatisfied. Setting $\text{comp}_j^\star \leftarrow \bar{\lambda}$ does not violate (INV), since predicates (6.3/6.4)and $(6.3/6.4)^-$ remain satisfied. Setting $\text{comp}_j^\star \leftarrow e$ does not improve the solution, since no predicate in (6.4), (6.7), $(6.3)^-$, (6.8), or (6.9) improves and predicate (6.10) becomes violated.

5. Let $\text{comp}_{j-1}^\star = \text{comp}_j^\star = \mu \in \{0, 1\}$. Then, predicate (6.3/6.4) is satisfied and setting $\text{comp}_j^\star \leftarrow \beta \in \{e, \bar{\mu}\}$ does not improve the solution, since predicate (6.3/6.4) becomes violated.

   Let us remark that the value of $\text{comp}_{2n}^\star$ is unknown. Note that for $j = 2n - 1$, the above case especially implies that if $\text{comp}_{2n}^\star = \bar{\mu}$ then this cannot lead to a change of $\text{comp}_{2n-1}^\star$ which improves the solution.

6. Let $\text{comp}_{j-1}^\star = \bar{\lambda}$ and $\text{comp}_j^\star = \lambda$. (INV) implies that $\text{comp}_{j+1}^\star = \lambda$ or $j = 2n-1$. Predicate (6.3/6.4) is unsatisfied and predicate $(6.3/6.4)^-$ is satisfied, in case $j < 2n - 1$. Setting $\text{comp}_j^\star \leftarrow \bar{\lambda}$ does not violate (INV). Now, consider setting $\text{comp}_j^\star \leftarrow e$. By definition, this is only possible for variables $\text{comp}_j^\star$ with $j < 2n - 1$. If j is odd, then predicate (6.5) is satisfied. Setting $\text{comp}_j^\star \leftarrow e$ does not improve the solution, since no predicate in (6.3) or (6.4) improves and predicate (6.8) becomes violated. If j is even, then setting $\text{comp}_j^\star \leftarrow e$ cannot improve predicate (6.7). The solution does not improve since no predicate in (6.4), (6.7), $(6.3)^-$, or (6.8) improves and predicate (6.9) becomes violated.

□

Lemma 6.8. (Φ, Ψ) *is a tight reduction.*

Proof. In this proof, we use the definition of $\mathscr{R}$ given in Definition 6.2. Note that by construction, all final assignments are contained in $\mathscr{R}$. Lemma 6.6 implies that

$\mathscr{R}$ contains all locally optimal solutions. Now, let $\mathsf{a} \in \mathscr{R}$. We consider an arbitrary sequence σ of improving steps and show for the first solution $\mathsf{a}' \in \mathscr{R}$ with $\mathsf{a}' \neq \mathsf{a}$ which is reached during the sequence, that either $\Psi(I, \mathsf{a}) = \Psi(I, \mathsf{a}')$ or $\Psi(I, \mathsf{a}')$ is a better neighbor of $\Psi(I, \mathsf{a})$. By definition, all predicates in (6.1) and (6.2) are satisfied in a and as they are the heaviest among the set of predicates, they remain satisfied throughout σ. First, consider the case $c_{\mathrm{C/F}}(\mathsf{a}(x^0), I) = c_{\mathrm{C/F}}(\mathsf{a}(x^1), I)$. Then, $\mathsf{a}(x^0) = \mathsf{a}(x^1)$ since we assumed that $c_{\mathrm{C/F}}(\mathsf{s}, I) \neq c_{\mathrm{C/F}}(\mathsf{s}', I)$ for solutions $\mathsf{s}, \mathsf{s}' \in F(I)$ with $\mathsf{s} \neq \mathsf{s}'$. The input vectors $\mathsf{a}(x^0)$ and $\mathsf{a}(x^1)$ cannot be changed in σ, since by (2a), $\mathsf{a}(x^{\bar{\lambda}}) = \textsc{Improve}(\mathsf{a}(x^{\lambda}), I)$ and thus $\mathsf{a}(y_i^{\bar{\lambda}}) \in \{d, \mathsf{a}(x^{\bar{\lambda}})\}$ for all $\lambda \in \{0,1\}$ and $i \in [n]$. This implies that $\Psi(I, \mathsf{a}') = \Psi(I, \mathsf{a}) = \mathsf{a}(x^0)$ for every solution a' in σ.

Now, let $c_{\mathrm{C/F}}(\mathsf{a}(x^{\bar{\lambda}}), I) > c_{\mathrm{C/F}}(\mathsf{a}(x^{\lambda}), I)$. We split σ into two phases. In a locally optimal solution $\mathsf{a}^\star \in F(\Phi(I))$, $\mathsf{a}^\star(x^0) = \mathsf{a}^\star(x^1)$ by Lemma 6.6. Thus, input vectors x^0 or x^1 have to change at least once.

Phase One: This Phase Continues Until Input Vector x^0 or x^1 Changes Since $Q_1(\mathsf{a}, \bar{\lambda}, \hat{k})$ is satisfied for some $\hat{k} \in [n]$ and by assumption on $\mathcal{S}$, there exists a path in $\mathcal{S}$ from input link $X_{\hat{k}}$ to all output links Z_i with $i \in [n]$, therefore $\mathsf{a}(z_i^{\bar{\lambda}}) = d$ for all $i \in [n]$. By assumption, property $Q_2(\mathsf{a}, \lambda)$ and all predicates in (6.2) are satisfied and therefore that $\mathsf{a}(z_i^{\lambda}) \in \{0,1\}$ for all $i \in [n]$. All predicates in (6.1) are satisfied and therefore that $\mathsf{a}(l) \in \{0,1\}$ for all $l \in \mathcal{X}_{\mathcal{L}}^{\lambda}$. Since $\mathsf{a} \in \mathscr{R}$, $\mathsf{a}(x^{\bar{\lambda}}) = \textsc{Improve}(\mathsf{a}(x^{\lambda}), I)$, there are no predicates which would now become satisfied by changing the value of some $x_i^{\bar{\lambda}}$ with $i \in [n]$. For some y_i^{λ} to change its binary value in the sequence σ, thus incentivizing changes to $x^{\bar{\lambda}}$, this requires that some x_i^{λ} with $i \in [n]$ changes its value to produce the new output. This implies that input vector x^{λ} changes first. Let $\mathsf{a}^\circ \in F(\Phi(I))$ be a solution which is reached at the end of phase one. There exists some $k \in [n]$ such that $\mathsf{a}^\circ(y_k^{\bar{\lambda}}) \in \{0,1\}$, $\mathsf{a}^\circ(y_k^{\bar{\lambda}}) \neq \mathsf{a}^\circ(x_k^{\lambda})$ and $\mathsf{a}^\circ(l) = d$ for all $l \in \mathcal{X}_{\mathrm{succ}(k)}^{\lambda}$; for all $i \in [n]$ and $l \in \mathcal{X}_{\mathrm{succ}(i)}^{\bar{\lambda}}$, $\mathsf{a}^\circ(l)$ takes its correct value in $\{0,1\}$. Thus, $Q_1(\mathsf{a}^\circ, \lambda, k)$ is satisfied.

Now, we show that $Q_2(\mathsf{a}^\circ, \bar{\lambda})$ is satisfied. Let σ_1 denote the subsequence of σ until the end of phase one. Because of Lemma 6.7, (INV) holds during σ_1. $Q_1(\mathsf{a}^\circ, \lambda, k)$ is satisfied and this implies that $\mathsf{a}^\circ(z_i^{\lambda}) = d$ for all $i \in [n]$. By Lemma 6.7, predicate $\mathsf{A}_{\mathrm{corr}(D_i)}$ in (6.2) is satisfied for all $i \in [n]$. Therefore, $\mathsf{a}^\circ(\mathrm{comp}_i) = \bar{\lambda}$ for all $i \in [n]$. Next, we show that $\mathsf{a}^\circ(\mathrm{comp}_2^\star) = \mathsf{a}^\circ(\mathrm{comp}_{2n-1}^\star) = \bar{\lambda}$. Together with (INV) this implies that $\mathsf{a}^\circ(\mathrm{comp}_j^\star) = \bar{\lambda}$ for all $j \in [2 : 2n-1]$. Therefore, $Q_2(\mathsf{a}^\circ, \bar{\lambda})$ is satisfied.

1. First, consider $\mathsf{a}^\circ(\mathrm{comp}_2^\star)$. By assumption, $Z(\mathsf{x})_1 = 0$ for all $\mathsf{x} \in F_{\mathrm{C/F}}(I)$ and therefore $\mathrm{pos}(\mathsf{a}) \geq 2$. (INV) implies that predicate $\mathsf{A}^3_{\mathrm{carry}(P_1)}$ in (6.7) is satisfied in assignment a° and therefore, $\mathsf{a}^\circ(\mathrm{comp}_2^\star) = \bar{\lambda}$.

2. Now, consider $\mathsf{a}^\circ(\mathrm{comp}_{2n-1}^\star)$. We want to show that $\mathsf{a}^\circ(\mathrm{comp}_{2n-1}^\star) = \bar{\lambda}$. Assume that $\mathsf{a}^\circ(\mathrm{comp}_{2n-1}^\star) = \lambda$. (INV) implies that $\mathsf{A}^2_{\mathrm{carry}(P_n)}$ in (6.5) is satisfied in assignment a° and therefore $\mathsf{a}^\circ(\mathrm{comp}_n) = \lambda$. A contradiction.

Phase Two: This Phase Starts with Changing the Value of x^λ and Terminates When the Entire Neighbor is Loaded For some $S \subseteq \{x_1^\lambda, \ldots, x_n^\lambda\}$, denote $\mathsf{a}(x^\lambda/S)$ the input vector when flipping all input bits $k \in S$. Let k_1 be the bit in which $\mathsf{a}(x^\lambda)$ and $\mathsf{a}(x^{\bar\lambda}) = \textsc{Improve}(\mathsf{a}(x^\lambda), I)$ differ and let k_2 be the bit in which $\mathsf{a}(x^{\bar\lambda})$ and $\textsc{Improve}(\mathsf{a}(x^{\bar\lambda}), I)$ differ. Phase one terminated by reaching a solution a° which satisfies $Q_1(\mathsf{a}^\circ, \lambda, k)$ for some $k \in \{k_1, k_2\}$ and $Q_2(\mathsf{a}^\circ, \bar\lambda)$. The input vectors are still unmodified, thus $\mathsf{a}(x^i) = \mathsf{a}^\circ(x^i)$ for all $i \in [0, 3]$.

Let setting $x_k^\lambda \leftarrow \mathsf{a}^\circ(y_k^{\bar\lambda})$ with $k \in \{k_1, k_2\}$, thus satisfying predicate (6.12) be the first change to $\mathsf{a}^\circ(x^\lambda)$; denote the resulting assignment a^1. By definition of I, $\mathsf{a}(x^\lambda/\{k_1\})$ is the best neighbor of $\mathsf{a}(x^\lambda)$ and this implies that $c_{\mathrm{C/F}}(\mathsf{a}(x^\lambda/\{k_1\}), I) \geq c_{\mathrm{C/F}}(\mathsf{a}(x^\lambda/\{k\}), I)$; hence, $\mathrm{pos}(\mathsf{a}^1) = n + 1$. Let σ' be an arbitrary sequence of improving steps without setting $x_{\bar k}^\lambda$ to a new value, where $\bar k \in \{k_1, k_2\} \setminus \{k\}$ and let $\mathsf{a}^\dagger \in \sigma'$ be a solution in σ'. Note that we are under the conditions of Lemma 6.7 when replacing a_0 by a^1, σ by σ', and λ by $\bar\lambda$. Lemma 6.7 now implies that $\mathsf{a}^\dagger(\mathrm{comp}^\star) = uw$, where $u \in e^*$, $w \in \bar\lambda^*$, and $\mathsf{a}^\dagger(\mathrm{comp}_i) \in \{e, \bar\lambda\}$ for all $i \in [n]$. All predicates $\mathsf{A}_{\mathrm{corr}(D_i)}$ in (6.2) are satisfied and therefore $\mathsf{a}^\dagger(z_i^{\bar\lambda}) \in \{0, 1\}$ for all $i \in [n]$. All predicates in (6.1) are satisfied as shown above and therefore $\mathsf{a}^\dagger(\ell) \in \{0, 1\}$ for all $\ell \in \mathcal{X}_{\mathcal{L}}^{\bar\lambda}$. Hence, input vector $\mathsf{a}^\dagger(x^{\bar\lambda})$ cannot be modified. No locally optimal solution can be reached this way and therefore $x_{\bar k}^\lambda$ is set to a new value in some solution during σ; denote the resulting solution a'. This requires that $Q_1(\mathsf{a}', \lambda, \bar k)$ is satisfied; with the same argumentation as in phase one, $Q_2(\mathsf{a}', \bar\lambda)$ is satisfied. Hence, we have reached a solution $\mathsf{a}' \in \mathscr{R}$ for which $\Psi(I, \mathsf{a}')$ is a better neighbor of $\Psi(I, \mathsf{a})$. □

Theorem 6.1. CIRCUIT/FLIP $\leq_{pls}$ $(3, 2, 3)$-MCA$_{3\text{-}par}$ *using a tight reduction.*

Proof. By Lemma 6.6, every locally optimal assignment A for $\Phi(I)$ is a final assignment. $\Psi(I, \mathsf{A})$ is locally optimal for I, since $x_i^0 = x_i^1$ holds for all $i \in [n]$ and this implies that $c_{\mathrm{C/F}}((x_i^\lambda)_{i=1}^n, I) = c_{\mathrm{C/F}}((y_i^{\bar\lambda})_{i=1}^n, I)$, where $\lambda = \mathrm{best}$. Thus, no improving flip of an input bit is possible for $\Psi(I, \mathsf{A})$.

Now, we show that the resulting set of constraints in our reduction is *3-partite.* We present a tri-coloring of all variables in length three predicates and a bi-coloring of all variables in length two predicates, using colors blue, red, and white. We slightly extend the construction described up to now in order to allow an easier coloring. First, consider the propagation trees in (6.11), (6.16), and (6.18). Each leaf can be colored independent of the colors of the other leaves. This may require to extend the tree by certain nodes of degree two. Next, consider the predicates (6.1) and (6.14) describing the correct work of the circuits. Recall that we assumed that every gate with three links is solely adjacent to gates with two links. Implanting gates with two links into some links allows us to color the predicates (6.1) and (6.14) with three colors independent of the colors given to the inputs and outputs x_i^0, x_i^1, y_i^0, y_i^1, z_i^0, z_i^1, x_i^2, x_i^3 for all $i \in [n]$ and best. We choose the following coloring: For all $i \in [n]$ and $j \in [2 : n]$, the variables x_i^0, x_i^1, z_i^0, best, and $\mathrm{comp}_{2i}^\star$ are colored white, the variables x_i^2, x_i^3, y_i^1, y_i^2, z_i^1, $\mathrm{comp}_{2j-1}^\star$, and b are colored blue. For all $i \in [n]$, the variable comp_i is colored red. We show that this coloring is a correct 3-coloring by giving the parameter list for

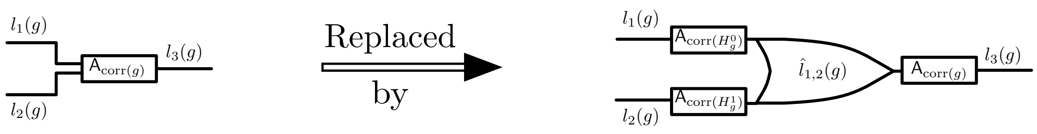

$l_1(g)$
$l_2(g)$
$\mathsf{A}_{\text{corr}(g)}$
$l_3(g)$
Replaced by
$l_1(g)$
$\mathsf{A}_{\text{corr}(H_g^0)}$
$l_2(g)$
$\mathsf{A}_{\text{corr}(H_g^1)}$
$\hat{l}_{1,2}(g)$
$\mathsf{A}_{\text{corr}(g)}$
$l_3(g)$

Figure 6.3: The modification for a predicate modeling a $(2,1)$-gate.

Identifier	Meaning
S^i	copy i of input circuit $\mathcal{S}$
$\mathcal{G}^i$	set of gates in circuit S^i
$\mathcal{G}^i_{\text{in}}$	set of gates incident to some input link in circuit S^i
$\mathcal{L}^i$	set of links in circuit S^i
$\mathcal{X}^j_{\mathcal{L}}$	set of hexary variables derived from links in $\mathcal{L}^j$ for $j \in \{0,1\}$
$\mathcal{X}^2_{\mathcal{L}}$	set of binary variables corresponding to links in $\mathcal{L}^2$
$\mathcal{X}^i_{\mathbb{L}}$	set of auxiliary link variables for $(2,1)$-gates in $\mathcal{G}^i$
$\tilde{\mathcal{Q}}$	$\mathcal{Q}^0 \cup \mathcal{Q}^1$, for all $\mathcal{Q} \in \{\mathcal{G}, \mathcal{L}, \mathcal{X}_{\mathcal{L}}, \mathcal{X}_{\mathbb{L}}\}$

Table 6.4: The labeling and meaning of identifiers we use, extending Table 6.1. Here, $i \in \{0,1,2\}$.

each of the remaining constraints. Variables that are leaves of propagation trees are denoted by dc (for "don't care"). In constraints (6.17) and (6.23), only one variable is not a don't care variable. For all $i \in [n]$, constraint D_i has variables $\text{comp}_i, z_i^0, z_i^1$; for all $i \in [2:n]$, constraint Q_i has variables $\text{comp}^\star_{2i-2}, \text{comp}^\star_{2i-1}$, dc; constraint P_1 has variables $\text{comp}_1, \text{comp}^\star_2$, dc; for all $i \in [2:n]$, constraint P_i has variables $\text{comp}^\star_{2i-1}, \text{comp}^\star_{2i}, \text{comp}_i$; constraint (6.12) has variables $x_i^{\bar{\kappa}}, y_i^{\kappa}$, dc; constraint (6.13) has variables x_i^{κ}, $x_i^{\kappa+2}$; constraint (6.15) has variables b, best, dc. □

6.4 $(2,3,6)$-MCA$_{\text{2-par}}$ is Tight $\mathcal{PLS}$-Complete

In this section, we prove that $(2,3,6)$-MCA$_{\text{2-par}}$ is tight $\mathcal{PLS}$-complete. We show this result by altering the reduction function Φ presented in Section 6.3. Conceptionally, we group two variables from a length three constraint as a pair and ensure consistency between a single variable and a grouped variable in a pair via auxiliary constraints. We have depicted this modification in Figure 6.3 for a length three predicate from (6.1) which simulates some $(2,1)$-gate $g \in \mathcal{G}$. Given an instance $I \in D_{\text{Circuit/Flip}}$, we construct an instance

$$\Phi^\star(I) = (\mathcal{F}^\star, \mathcal{X}^\star) \in D_{(2,3,6)\text{-MCA}_{\text{2-par}}}$$

with a set of constraints $\mathcal{F}^\star$ and a set of at most 6-valued variables $\mathcal{X}^\star$. Every constraint $C_i \in \mathcal{F}^\star$ has length at most 2 and every variable $x \in \mathcal{X}^\star$ appears in at most 3 constraints.

6.4.1 The Set of Constraints

Whenever we use κ or λ when listing constraints, then this is to denote that there are actually predicates in $\mathcal{F}^\star$ for all $\kappa, \lambda \in \{0,1\}$. Constraints consisting of more than one predicate are listed in Table 6.5. All other constraints are weighted predicates. Eventually, the *set of constraints* is

$$\begin{aligned}\mathcal{F}^\star := \;& \tilde{\mathcal{G}} \cup \mathcal{G}^2 \cup \\ & \{g_\lambda \mid g \in \tilde{\mathcal{G}} \cup \mathcal{G}^2 \text{ a } (2,1)\text{-gate}\} \cup \{g_\lambda \mid g \in \tilde{\mathcal{G}} \cup \mathcal{G}^2 \text{ a } (1,2)\text{-gate}\} \cup \\ & \{\mathsf{A}^\kappa_{\text{join}_i}, \mathsf{B}^\kappa_{\text{load}_i}, \mathsf{B}^\kappa_{\text{copy}_i} D_i, P_i, Q^\kappa_i, \mid i \in [n]\} \cup \{\mathsf{B}_{\text{propagate}}\} \cup T_1 \cup T_2,\end{aligned}$$

where T_1 and T_2 denote the sets of predicates for the respective propagation trees (see next paragraph for details).

6.4.2 The Set of Variables

We introduce the *circuit variables* listed in the upper section of Table 6.6 and the *variables for comparison* listed in the lower section of Table 6.6. Some of the variables we use in this reduction are cartesian products of variables from Section 6.3; confer the next paragraph for general similarities and differences to our reduction in Section 6.3. We additionally introduce link variables with special names $\hat{x}^\kappa_i$, $\hat{y}^\kappa_i$, and $\hat{z}^\kappa_i$ in $\mathcal{X}^\kappa_{\mathcal{L}}$ and $x^{\kappa+2}_i$ in $\mathcal{X}^2_{\mathcal{L}}$ for all $i \in [n]$. We also outlined the interpretation of the components and their relation to variables from Section 6.3, where possible. Here, b_l denotes a binary variable which is introduced for each link $l \in \tilde{\mathcal{L}} \setminus \{X^0_i, X^1_i \mid i \in [n]\}$ in this reduction. Variables b_l correspond to binary variables b_g from Section 6.3 in the sense that now each link variable $l \in \tilde{\mathcal{X}}_{\mathcal{L}}$ carries its separate b_l variable. Note that, $P_2(\hat{k}_i)$ is binary for all $i \in [n]$ in order to keep the valence of $\hat{k}_i$ at six, whereas variables c_i with $i \in [n-1]$ are defined as ternary variables in Section 6.3. We compensate for this with additional predicates. Additional variables in tree T_1 propagate the value of $P_1(\hat{k}_{n+1})$ to all $P_2(\hat{x}^{\kappa+2}_i)$ with $i \in [n]$. Additional variables in tree T_2 propagate the value of $P_2(\hat{k}_1)$ to all variables $P_2(\hat{l})$, which are not input links of $(1,2)$-gates. Eventually, the *set of variables* is now

$$\begin{aligned}\mathcal{X}^\star := \;& \tilde{\mathcal{X}}_{\mathcal{L}} \cup \tilde{\mathcal{X}}_{\mathbb{L}} \cup \mathcal{X}^2_{\mathcal{L}} \cup \mathcal{X}^2_{\mathbb{L}} \cup \\ & \{\hat{z}_i, \hat{x}^{\kappa+2}_i \mid i \in [n]\} \cup \{\hat{k}_i \mid i \in [n+1]\} \cup \{\text{best}\} \cup V_{T_1} \cup V_{T_2},\end{aligned}$$

where V_{T_i} with $i \in \{1,2\}$ denotes the set of variables in propagation tree T_i.

6.4.3 Similarities and Differences to our Reduction in Section 6.3

While our new construction still follows the general method presented in Section 6.2, the actual *implementation with constraints differs in* the representation of the involved *circuits*, the *comparator*, the *loading* and *steering logic*. These modifications have an effect on the set of *constraints*, as well as the underlying set of *variables*.

Constraint,	Index	Set of weighted predicates	Identifiers
D_i,	$i \in [n]$	$\mathsf{A}_{\text{corr}(D_i)}, \mathsf{A}_{\text{out}(D_i)}$	(6.2*), (6.6*)
P_i,	$i \in [n]$	$\mathsf{A}_{\text{carry}(P_i)}, \mathsf{A}_{\text{care}(P_i)}, \mathsf{B}_{\text{copy}(P_i)}, \mathsf{C}^0_{\text{tighten}(P_i)}$, $\mathsf{C}^0_{\text{relax}(P_i)}, \mathsf{C}^1_{\text{tighten}(P_i)}, \mathsf{C}^2_{\text{tighten}(P_i)}, \mathsf{C}^2_{\text{relax}(P_i)}$	(6.7*), (6.8*), (6.16*), (6.19a*), (6.19b*), (6.19c*), (6.19d*), (6.19e*)
Q^κ_i,	$i \in [n]$	$\mathsf{B}_{\text{join}(Q^\kappa_i)}, \mathsf{B}_{\text{copy}(Q^\kappa_i)}$	(6.11b*), (6.13a*)
g,	$g \in \tilde{\mathcal{G}} \setminus \tilde{\mathcal{G}}_{\text{in}}$, a $(1,1)$-gate	$\mathsf{A}_{\text{corr}(g)}, \mathsf{C}^1_{\text{value}(g)}$	(6.1a*), (6.23c*)
g_λ,	$g \in \tilde{\mathcal{G}}$, a $(1,2)$-gate	$\mathsf{A}_{\text{corr}(g_\lambda)}, \mathsf{B}_{\text{copy}(g_\lambda)}, \mathsf{C}^1_{\text{value}(g_\lambda)}$	(6.1a*), (6.18b*), (6.23c*)
g,	$g \in \tilde{\mathcal{G}}$, a $(2,1)$-gate	$\mathsf{A}_{\text{corr}(g)}, \mathsf{C}^0_{\text{value}(g)}, \mathsf{C}^1_{\text{value}(g)}$	(6.1b*), (6.23b*), (6.23c*)

Table 6.5: Constraints consisting of more than one weighted predicate where $\lambda, \kappa \in \{0,1\}$

Variable	Index	Domain	Constraint	Interpretation of the Tuple
$\hat{l}$	$l \in \tilde{\mathcal{L}} \setminus \{X_i^0, X_i^1 \mid i \in [n]\}$	$\{0,1,d\} \times \{0,1\}$		$\hat{l} = (l, b_l)$
	if inp. link for a $(1,1)$-gate		g, g', $N^2(\hat{l})$	
	if inp. link for a $(1,2)$-gate		g, g'_0 g'_1	
	if inp. link $\hat{l}_\lambda$ for a $(2,1)$-gate		g, g'_λ, $N^2(\hat{l})$	
$\hat{x}_i^\kappa$	$i \in [n]$	$\{0,1\} \times \{0,1\}$	g, Q_i^κ, $\mathsf{B}_{\text{load}_i}^{\bar{\kappa}}$	$\hat{x}_i^\kappa = (x_i^\kappa, a_i^\kappa)$
$\hat{y}_i^\kappa$	$i \in [n]$	$\{0,1,d\} \times \{0,1\}$	g, $\mathsf{B}_{\text{load}_i}^\kappa$, $N^2(\hat{y}_i^\kappa)$	$\hat{y}_i^\kappa = (y_i^\kappa, b_{y_i^\kappa})$
$\hat{z}_i^\kappa$	$i \in [n]$	$\{0,1,d\} \times \{0,1\}$	g, $\mathsf{A}_{\text{join}_i}^\kappa$, $N^2(\hat{z}_i^\kappa)$	$\hat{z}_i^\kappa = (z_i^\kappa, b_{z_i^\kappa})$
$\hat{z}_i$	$i \in [n]$	$\{0,1\} \times \{0,1,e\}$	$\mathsf{A}_{\text{join}_i}^\kappa$, D_i	
$\hat{l}_{1,2}(g)$	$g \in \tilde{\mathcal{G}}$ a $(2,1)$-gate	$\{\{0,1\} \times \{0,1\}\} \cup d$	g_κ, g	$\hat{l}_{1,2}(g) = (l_1(g), l_2(g))$
				if $\hat{l}_{1,2}(g) \in \{0,1\}^2$
$\hat{x}_i^{\kappa+2}$	$i \in [n]$	$\{0,1\} \times \{0,1\}$	$N^1(\hat{x}_i^{\kappa+2})$, Q_i^κ, $\mathsf{B}_{\text{copy}_i}^\kappa$	$\hat{x}_i^{\kappa+2} = (x_i^{\kappa+2}, a_i^\kappa)$
l	$l \in \mathcal{X}_{\mathcal{L}}^2 \setminus \{\{\text{best}\}\} \cup \{x_i^{\kappa+2} \mid i \in [n]\}\}$	$\{0,1\}$		
	if inp. link for a $(1,1)$-gate		g, g'	
	if inp. link for a $(1,2)$-gate		g, g_0, g_1	
	if inp. link $\hat{l}_\lambda$ for a $(2,1)$-gate		g, g_λ	
$\hat{l}_{1,2}(g)$	$g \in \mathcal{G}^2$ a $(2,1)$-gate	$\{0,1\} \times \{0,1\}$	g_κ, g	$\hat{l}_{1,2}(g) = (l_1(g), l_2(g))$
$x_i^{\kappa+2}$	$i \in [n]$	$\{0,1\}$	$\mathsf{B}_{\text{copy}_i}^\kappa$, g	
best		$\{0,1,e\}$	g, $\mathsf{B}_{\text{propagate}}$	
$\hat{k}_1$		$\{0,1,e\} \times \{0,1\}$	$D_1, P_1, N^2(\hat{k}_1)$	$\hat{k}_i = (\text{comp}_1, c_1)$
$\hat{k}_i$	$i \in [2 : n-2]$	$\{0,1,e\} \times \{0,1\}$	P_{i-1}, D_i, P_i	$\hat{k}_i = (\text{comp}_i, c_i)$
$\hat{k}_i$	$i \in \{n-1, n\}$	$\{0,1\} \times \{0,1\}$	P_{i-1}, D_i, P_i	$\hat{k}_i = (\text{comp}_i, c_i)$
$\hat{k}_{n+1}$		$\{0,1\} \times \{0,1\}$	P_n, $N^1(\hat{k}_{n+1})$, $\mathsf{B}_{\text{propagate}}$	$\hat{k}_{n+1} = (\text{comp}_n, c_n)$

Table 6.6: The set of variables, their respective domains, and constraints for $\kappa, \lambda \in \{0,1\}$. Here, constraints g and g' refer to gates with the respective input or output variable; $N^i(x)$ denotes the constraint containing variable $x \in \mathcal{X}^\star$ in propagation tree T_i with $i \in \{1,2\}$.

In this reduction, every constraint can only have length at most two which affects the set of predicates for *circuits* S^0, S^1, and S^2. Note that the description of $(1,1)$- and $(1,2)$-gates in Section 6.3 already contains predicates of length two where each variable appears at most three times. For $(2,1)$-gates, we have to change the corresponding predicates. For each $(2,1)$-gates, we first introduce auxiliary constraints that pool the two input values in a single variable taking five values and then simulate the actual gate in a predicate using the variable taking five values and the corresponding output variable. Hence, we have to alter the set of predicates in (6.1). In Section 6.3, we were able to feed the output links of both circuits directly into the comparator. This approach is not suitable for our construction, where each constraint is limited to length at most two. Instead, we introduce new variables $\hat{z}_i$ for all $i \in [n]$ which take six values. Here, the first component is binary and the second component is ternary. The design principle is such that if the second component is e, then both circuits output the same binary value stored in the first component; otherwise the first component holds the binary value of the circuit the second component points to. In order to implement this, we introduce a new predicate in (6.1c*) for each output variable z_i^κ for all $i \in [n]$ and $\kappa \in \{0,1\}$.

The *comparator* in this construction requires no auxiliary variables $\text{comp}_j^\star$ for all $j \in [2:2n]$ as in Section 6.3 and manages to solely work with variables $\hat{k}_i$ for $i \in [n]$ which are cartesian products of the respective variables comp_i and binary variables c_i. The newly introduced variables $\hat{z}_i$ take their toll on our construction, as we have to change the crucial predicates in (6.2) and (6.6). The most important change is that predicates (6.2*) can also be satisfied in dependence on the output, in case the comparator is binary. Additional, we introduce small incentives in predicates (6.19a*)–(6.19e*) which supersede predicates (6.19)–(6.22) to nudge the comparator, as well as the values of the circuits in the right direction.

Additionally, the *loading logic* has to be adapted to fit the requirements of the reduction. Now, variables a_i^κ for all $i \in [n], \kappa \in \{0,1\}$ from Section 6.3 are passed to a component of newly introduced auxiliary variables $\hat{x}_i^{\kappa+2}$ and on to a component of variables $\hat{x}_i^\kappa$. Passing the value of x_i^κ to $x_i^{\kappa+2}$ is then done in the reverse direction, using the same constraints. This way, we are able to ensure that each variable appears in at most three constraints. We use predicates (6.11a*)–(6.13b*) for this construction.

Now, consider the *steering logic*. Propagation tree T_3 is removed from the construction, as we propagate the $c_{\dots}$ values from Section 6.3 within the comparator, using the structure of the constraints in predicates in (6.16*). There are in principle two possibilities to set the b_l variables. One is to set these variables via additional variables using e.g. a propagation tree, as done in Section 6.3. The second method is to propagate the value within the constraints which model the given circuit, using no additional variables. The first method increments the appearance of the respective variable by one, since the respective variable is already contained in a constraint of length two. So, this method is not suitable for input variables of $(1,2)$-gates, which already appear three times. The second method requires that each link variable stores its own b_l variable. By construction, $\hat{l}_{1,2}(g)$ variables from $(2,1)$-gates $g \in \tilde{\mathcal{G}}$ do not possess such a b_l variable; hence, this approach is not suitable for these variables. To overcome both obstacles, we choose a *hybrid approach*, where we use a propagation tree to set the b_l variables for all link variables $l \in \tilde{\mathcal{X}_\mathcal{L}}$ which are not input links of $(1,2)$-gates in predicates (6.18a*). The b_l variables of input links of $(1,2)$-gates are then set by using the b_l variables of incident links in predicates (6.18b*). Setting $\hat{l}_{1,2}(g)$ variables from $(2,1)$-gates $g \in \tilde{\mathcal{G}}$ to binary values or d is controlled using the b_l variables of incident links in predicates (6.23b*).

6.4.4 The Modified Constraint-Graph of Our Reduction

With these modifications, the *constraint-graph* from Section 6.3 is now a *hypergraph* $\mathcal{H}$. Here, nodes are constraints from $\mathcal{F}^\star$ and hyperedges are defined by variables in $\mathcal{X}^\star$. Each hyperedge consists of two or three nodes, corresponding to the fact that each variable occurs in at most three constraints. Note that the definition of $\mathsf{R}(\ell, (x_1, \ldots, x_n))$ naturally extends to hyperedges ℓ. The topological sorting also carries over to the hypergraph $\mathcal{H}$.

6.4.5 The Set of Modified Predicates

We modify the set of predicates presented in Section 6.3. Predicates labeled (X) from Section 6.3 which are modified, carry the identifying label $(X^\star)$. We preserve the original hierarchical structure of the weights. Let us remark that some reused predicates depend on three arguments, two of which are projections of the same variable.

Level I: Computing and Comparing the Output

For all $(1,2)$-gates $g \in \tilde{\mathcal{G}}$, we replace predicates (6.1) with two predicates

$$\begin{aligned}&\mathsf{A}_{\mathrm{corr}(g_\lambda)}(\hat{l}_1(g), \hat{l}_{\lambda+2}(g)) = \\ &\qquad [P_1(\hat{l}_{\lambda+2}(g)) = d \mid \\ &\qquad P_1(\hat{l}_1(g)), P_1(\hat{l}_{\lambda+2}(g)) \in \{0,1\} \wedge \textsc{Correct}_g(P_1(\hat{l}_1(g)), P_1(\hat{l}_{\lambda+2}(g)))].\end{aligned} \tag{6.1a*}$$

For all $(1,1)$-gates $g \in \tilde{\mathcal{G}}$ with input link $\hat{l}_1(g)$ and output link $\hat{l}_2(g)$, we replace the existing predicate with a single predicate $\mathsf{A}_{\mathrm{corr}(g)}(\hat{l}_1(g), \hat{l}_2(g))$, as defined in predicates (6.1a*). Every predicate in (6.1) for a $(2,1)$-gate $g \in \tilde{\mathcal{G}}$ is replaced by three predicates

$$\begin{aligned}&\mathsf{A}_{\mathrm{corr}(g_\lambda)}(\hat{l}_{\lambda+1}(g), \hat{l}_{1,2}(g)) =\\ &\qquad[\hat{l}_{1,2}(g) = d \mid\\ &\qquad\hat{l}_{1,2}(g) \in \{0,1\}^2 \wedge P_{\lambda+1}(\hat{l}_{1,2}(g)) = P_1(\hat{l}_{\lambda+1}(g))]\end{aligned} \tag{6.1a*}$$

$$\begin{aligned}&\mathsf{A}_{\mathrm{corr}(g)}(\hat{l}_{1,2}(g), \hat{l}_3(g)) =\\ &\qquad[\hat{l}_{1,2}(g) = d \Rightarrow P_1(\hat{l}_3(g)) = d] \text{ and}\\ &\qquad[\hat{l}_{1,2}(g) \in \{0,1\}^2 \wedge P_1(\hat{l}_3(g)) \in \{0,1\}\\ &\qquad\quad\Rightarrow \textsc{Correct}_g(P_1(\hat{l}_{1,2}(g)), P_2(\hat{l}_{1,2}(g)), P_1(\hat{l}_3(g)))].\end{aligned} \tag{6.1b*}$$

We introduce two predicates which join the values of $P_1(\hat{z}_i^\kappa)$ in $\hat{z}_i$, with weights decreasing in i

$$\mathsf{A}^\kappa_{\mathrm{join}_i}(\hat{z}_i^\kappa, \hat{z}_i) = [P_2(\hat{z}_i) = \bar{\kappa} \vee P_1(\hat{z}_i) = P_1(\hat{z}_i^\kappa)]. \tag{6.1c*}$$

Recall that $P_1(\hat{z}_i) \in \{0,1\}$ by definition. All predicates in (6.2)–(6.7) are replaced by the following length two predicates (6.2*), (6.6*), and (6.7*), which are in interleaving order $\dots$, $\mathsf{A}_{\mathrm{corr}(D_i)}$, $\mathsf{A}_{\mathrm{out}(D_i)}$, $\mathsf{A}_{\mathrm{carry}(P_i)}$, $\mathsf{A}_{\mathrm{corr}(D_{i+1})}$, $\mathsf{A}_{\mathrm{out}(D_{i+1})}$, $\mathsf{A}_{\mathrm{carry}(P_{i+1})}$, $\dots$ for all $i \in [n]$ and weight decrease in this order. For all $i \in [n]$, we replace each predicate in (6.2) with predicate

$$\begin{aligned}&\mathsf{A}_{\mathrm{corr}(D_i)}(\hat{z}_i, \hat{k}_i) =\\ &\qquad[P_1(\hat{k}_i) = P_2(\hat{z}_i) = e \wedge i < n-1 \mid\\ &\qquad P_1(\hat{k}_i) \in \{0,1\} \wedge [P_2(\hat{z}_i) \in \{e, P_1(\hat{k}_i)\} \vee P_1(\hat{z}_i) = 1]].\end{aligned} \tag{6.2*}$$

and each predicate in (6.6) with predicate

$$\mathsf{A}_{\mathrm{out}(D_i)}(\hat{z}_i, \hat{k}_i) = [P_2(\hat{z}_i) \in \{e, P_1(\hat{k}_i)\} \wedge P_1(\hat{z}_i) = 1]. \tag{6.6*}$$

and each predicate in (6.7) with predicate

$$\begin{aligned}\mathsf{A}_{\mathrm{carry}(P_i)}(\hat{k}_i, \hat{k}_{i+1}) = [&P_1(\hat{k}_i) = e \wedge i < n-1 \mid\\ &P_1(\hat{k}_i) = P_1(\hat{k}_{i+1}) \in \{0,1\}].\end{aligned} \tag{6.7*}$$

All predicates in (6.8)–(6.10) are removed from the list of predicates. Instead, we introduce predicates for all $i \in [n]$ with weight increasing in i

$$\mathsf{A}_{\mathrm{care}(P_i)}(\hat{k}_i, \hat{k}_{i+1}) = [(P_1(\hat{k}_i) = e \wedge i < n-1) \vee P_1(\hat{k}_{i+1}) \in \{0,1\}]. \tag{6.8*}$$

All predicates in (6.11) are replaced by predicates which propagate the value of $P_1(\hat{k}_{n+1})$ to all variables $P_2(\hat{x}_i^{\kappa+2})$ with $i \in [n]$, using propagation tree T_1, where

$$\begin{aligned}\mathrm{root}(T_1) &= P_1(\hat{k}_{n+1});\\ \mathrm{leaves}(T_1) &= \{P_2(\hat{x}_i^{\kappa+2}) \mid i \in [n]\}.\end{aligned} \tag{6.11a*}$$

For all $i \in [n]$, we propagate the value of $P_2(\hat{x}_i^{\kappa+2})$ to $P_2(\hat{x}_i^\kappa)$, where weights decrease in i in predicates

$$\mathsf{B}_{\mathrm{join}(Q_i^\kappa)}(\hat{x}_i^{\kappa+2}, \hat{x}_i^\kappa) = [P_2(\hat{x}_i^{\kappa+2}) = P_2(\hat{x}_i^\kappa)]. \tag{6.11b*}$$

Level II: Loading a Better Neighbor and Pushing the Comparator

Predicates in (6.12) remain unchanged

$$\mathsf{B}^{\kappa}_{\mathrm{load}_i}(\hat{x}^{\bar{\kappa}}_i, \hat{y}^{\kappa}_i) = [P_1(\hat{x}^{\bar{\kappa}}_i) = P_1(\hat{y}^{\kappa}_i) \wedge P_2(\hat{x}^{\bar{\kappa}}_i) = \kappa]. \tag{6.12*}$$

All predicates in (6.13) are replaced by predicates of weight increasing in i which copy the value of $P_1(\hat{x}^{\kappa}_i)$ to $x^{\kappa+2}_i$ in predicates

$$\mathsf{B}_{\mathrm{copy}(Q^{\kappa}_i)}(\hat{x}^{\kappa}_i, \hat{x}^{\kappa+2}_i) = [P_1(\hat{x}^{\kappa}_i) = P_1(\hat{x}^{\kappa+2}_i)] \tag{6.13a*}$$

$$\mathsf{B}^{\kappa}_{\mathrm{copy}_i}(\hat{x}^{\kappa+2}_i, x^{\kappa+2}_i) = [P_1(\hat{x}^{\kappa+2}_i) = x^{\kappa+2}_i]. \tag{6.13b*}$$

For all $(2,1)$-gates $g \in \mathcal{G}^2$, we replace all predicates in (6.14) by two predicates

$$\mathsf{B}_{\mathrm{corr}(g_\lambda)}(l_{\lambda+1}(g), \hat{l}_{1,2}(g)) = [P_{\lambda+1}(\hat{l}_{1,2}(g)) = l_{\lambda+1}(g)] \tag{6.14a*}$$

$$\mathsf{B}_{\mathrm{corr}(g)}(\hat{l}_{1,2}(g), l_3(g)) = [\textsc{Correct}_g(P_1(\hat{l}_{1,2}(g)), P_2(\hat{l}_{1,2}(g)), l_3(g))]. \tag{6.14b*}$$

For all $(1,2)$-gates $g \in \mathcal{G}^2$, we replace all predicates in (6.14) with two predicates

$$\mathsf{B}_{\mathrm{corr}(g_\lambda)}(l_1(g), l_{\lambda+2}(g)) = [\textsc{Correct}_g(l_1(g), l_{\lambda+2}(g))]. \tag{6.14a*}$$

For all $(1,1)$-gates $g \in \mathcal{G}^2$ with input link $l_1(g)$ and output link $l_2(g)$, we replace the existing predicate (6.14) with a single predicate $\mathsf{B}_{\mathrm{corr}(g)}(l_1(g), l_2(g))$, as defined in predicates (6.14a*). Recall that in the single output link best, S^2 returns the index of the input vector which yields the larger binary output or e if both input vectors yield the identical binary output. Predicate (6.15) is replaced by predicate

$$\begin{aligned}\mathsf{B}_{\mathrm{propagate}}(\hat{k}_{n+1}, \mathrm{best}) = \\ [\mathrm{best} = e \wedge P_1(\hat{k}_{n+1}) = P_2(\hat{k}_{n+1}) \mid \\ \mathrm{best} = P_2(\hat{k}_{n+1})].\end{aligned} \tag{6.15*}$$

All predicates in (6.16)–(6.18) are replaced by the following predicates: For all $i \in [n]$, we introduce predicates which propagate the values of $P_2(\hat{k}_{i+1})$ to $P_2(\hat{k}_i)$ in

$$\mathsf{B}_{\mathrm{copy}(P_i)}(\hat{k}_i, \hat{k}_{i+1}) = [P_2(\hat{k}_i) = P_2(\hat{k}_{i+1})] \tag{6.16*}$$

and weights increase in i. Predicate (6.17) is removed from the list of predicates. The value of $P_2(\hat{k}_1)$ is copied to all $P_2(\hat{l})$ with $\hat{l} \in \tilde{\mathcal{X}}_{\mathcal{L}} \setminus \{\hat{x}^{\kappa}_i \mid i \in [n]\}$, which are *not* input links of $(1,2)$-gates, using propagation tree T_2

$$\begin{aligned}\mathrm{root}(T_2) &= P_2(\hat{k}_1);\\ \mathrm{leaves}(T_2) &= \{P_2(\hat{l}) \mid \hat{l} \in \tilde{\mathcal{X}}_{\mathcal{L}} \setminus \{\hat{x}^{\kappa}_i \mid i \in [n]\} \text{ not inp. link of a } (1,2)\text{-gate}\}.\end{aligned} \tag{6.18a*}$$

For all $\hat{l}_1(g) \in \tilde{\mathcal{X}}_{\mathcal{L}} \setminus \{\hat{x}^{\kappa}_i \mid i \in [n]\}$ which are input links for $(1,2)$-gates with input link $\hat{l}_1(g)$ and output links $\hat{l}_2(g), \hat{l}_3(g)$, we propagate the value of $P_2(\hat{l}_2(g))$ and $P_2(\hat{l}_3(g))$ to $P_2(\hat{l}_1(g))$ in predicates

$$\mathsf{B}_{\mathrm{copy}(g_\lambda)}(\hat{l}_1(g), \hat{l}_{\lambda+2}(g)) = [P_2(\hat{l}_1(g)) = P_2(\hat{l}_{\lambda+2}(g))]. \tag{6.18b*}$$

Weights decrease in the topological order of the gates and in λ within the gates.

Level III: Small Incentives

Predicates in (6.19)–(6.22) are removed from the list of predicates. Instead, we introduce predicates for all $i \in [n]$ with weight increasing in i in

$$\mathsf{C}^0_{\text{tighten}(P_i)}(\hat{k}_i, \hat{z}_i) = [P_2(\hat{z}_i) = P_2(\hat{k}_i) = P_1(\hat{k}_i)] \tag{6.19a*}$$

$$\mathsf{C}^0_{\text{relax}(P_i)}(\hat{z}_i) = [P_2(\hat{z}_i) = e] \tag{6.19b*}$$

$$\mathsf{C}^1_{\text{tighten}(P_i)}(\hat{k}_i, \hat{z}_i) = [P_2(\hat{z}_i) = P_2(\hat{k}_i)] \tag{6.19c*}$$

$$\mathsf{C}^2_{\text{tighten}(P_i)}(\hat{k}_i) = [P_1(\hat{k}_i) = P_2(\hat{k}_i)] \tag{6.19d*}$$

$$\mathsf{C}^2_{\text{relax}(P_i)}(\hat{k}_i) = [P_1(\hat{k}_i) = e]. \tag{6.19e*}$$

Predicates in (6.23) are removed from the list of predicates. For all link variables $\hat{l}_{1,2}(g) \in \mathcal{X}^\kappa_{\mathbb{L}}$ from $(2,1)$-gates with output link $\hat{l}_3(g) \in \mathcal{X}^\kappa_{\mathcal{L}}$, we introduce predicates

$$\begin{aligned}\mathsf{C}^0_{\text{value}(g)}(\hat{l}_{1,2}(g), \hat{l}_3(g)) = \\ [P_2(\hat{l}_3(g)) = \kappa \wedge \hat{l}_{1,2}(g) \neq d \mid \\ P_2(\hat{l}_3(g)) = \bar{\kappa} \wedge \hat{l}_{1,2}(g) = d];\end{aligned} \tag{6.23b*}$$

for all link variables $\hat{l}(g) \in \mathcal{X}^\kappa_{\mathcal{L}} \setminus \{x^\kappa_i \mid i \in [n]\}$, we introduce predicates

$$\begin{aligned}\mathsf{C}^1_{\text{value}(g)}(\hat{l}(g)) = \\ [P_2(\hat{l}(g)) = \kappa \wedge P_1(\hat{l}(g)) \neq d \mid \\ P_2(\hat{l}(g)) = \bar{\kappa} \wedge P_1(\hat{l}(g)) = d].\end{aligned} \tag{6.23c*}$$

Again, weights increase in the topological order of the gates.

Solution Mapping

Let $\mathsf{a} \in F(\Phi^\star(I))$ and for $\lambda \in \{0,1\}$ denote

$$\mathsf{a}(x^\lambda) := \mathsf{a}(P_1(\hat{x}^\lambda_1)), \ldots, \mathsf{a}(P_1(\hat{x}^\lambda_n)).$$

We drop the assignment, where it is clear from the context. Similar to Section 6.3, function $\Psi^\star(I, \mathsf{a})$ returns $\mathsf{a}(x^\lambda)$ for $\lambda \in \{0,1\}$ if $c_{\text{C/F}}(\mathsf{a}(x^\lambda), I) > c_{\text{C/F}}(\mathsf{a}(x^{\bar{\lambda}}), I)$ and $\mathsf{a}(x^0)$ otherwise. This terminates the description of the reduction. ○

6.4.6 Proving the Correctness and Tightness of the Reduction

The proof of correctness and tightness of the reduction given above follows the proof of Theorem 6.1 in Section 6.3, as we again show *properties of a locally optimal solution* $\mathsf{a} \in F(\Phi^\star(I))$.

Roadmap of the Proof In Lemma 6.9, we first focus on the set of predicates which are trivially satisfied in a and derive properties for the involved variables in Corollary 6.2. In Lemma 6.10, we show the connection between $P_1(\hat{k}_i)$ and $P_2(\hat{k}_{n+1})$ for all $i \in [n+1]$. In Lemma 6.11, we show that for $\lambda = P_2(\hat{k}_{n+1})$, all link variables in circuit $S^{\bar{\lambda}}$ are set to d; by Corollary 6.2, all link variables in circuit S^{λ} are set to binary values. In Lemma 6.12, we prove that best $= e$. We close our proof of correctness by explicitly stating the variable assignment in a in Definition 6.3 and show that any deviating assignment cannot be locally optimal in Lemma 6.13. To consider tightness, we present the set $\mathscr{R}^\star$ in Definition 6.4. In Lemma 6.14, we prove a structural invariant for the vector of variables $P_1(\hat{k}_i)$ for all $i \in [n]$ in any sequence of improving steps which starts in $\mathscr{R}^\star$ and fulfills some additional properties which are useful in the prove of tightness of our reduction in Lemma 6.15. Theorem 6.2 then shows that $(2,3,6)$-MCA$_{2\text{-par}}$ is tight $\mathcal{PLS}$-complete.

Lemma 6.9. *All predicates in* (6.1a*)*,* (6.1b*)*,* (6.7*)*–*(6.11b*)*, and* (6.13a*)*–*(6.18a*) *are satisfied.*

Proof. First, consider predicates (6.11a*) and (6.18a*) for propagation trees. The value of each child node appearing for the first time can be set to the value of the parent node and only predicates of lower weight are violated. Now, consider the remaining predicates, except for predicates in (6.8*). Note that in each of these predicates, at least one variable or projection of a variable appears for the first time, with respect to the given order. In case of the propagation trees, all variables except the root occur for the first time. In detail, $P_1(\hat{l}_2(g))$ in (6.1a*) for all $(1,1)$-gates $g \in \tilde{\mathcal{G}}$, $P_1(\hat{l}_{\lambda+2}(g))$ in (6.1a*) for all $(1,2)$-gates $g \in \tilde{\mathcal{G}}, \lambda \in \{0,1\}$, and $\hat{l}_{1,2}(g)$ in (6.1a*), $P_1(\hat{l}_3(g))$ in (6.1b*), for all $(2,1)$-gates $g \in \tilde{\mathcal{G}}$. Furthermore, $P_1(\hat{k}_{i+1})$ in (6.7*), $P_2(\hat{x}_i^\kappa)$ in (6.11b*), $P_1(\hat{x}_i^{\kappa+2})$ in (6.13a*), $x_i^{\kappa+2}$ in (6.13b*), $P_2(\hat{k}_{n+1})$ in (6.15*), and $P_2(\hat{k}_i)$ in (6.16*), for all $i \in [n]$ and $\kappa \in \{0,1\}$. Finally, $l_2(g)$ in (6.14a*) for all $(1,1)$-gates $g \in \mathcal{G}^2$, $l_{\lambda+2}(g)$ in (6.14a*) for all $(1,2)$-gates $g \in \mathcal{G}^2$, $P_{\lambda+1}(\hat{l}_{1,2}(g))$ in (6.14a*), and $l_3(g)$ in (6.14b*) for all $(2,1)$-gates $g \in \mathcal{G}^2, \lambda \in \{0,1\}$. Thus, the value of a variable or a projection of a variable appearing for the first time can be set to satisfy the predicate and only predicates of lower weight are violated. As shown above, all predicates in (6.7*) are satisfied and this implies that all predicates in (6.8*) are satisfied. □

Corollary 6.2. *The following properties hold:*

1. *For all $i \in [n]$ and $\kappa \in \{0,1\}$, $P_1(\hat{x}_i^\kappa) = P_1(\hat{x}_i^{\kappa+2}) = x_i^{\kappa+2}$.*
2. *For all $\hat{l} \in \mathcal{X}_{\mathcal{L}}^\kappa$ and $\kappa \in \{0,1\}$, $P_1(\hat{l}) \in \{d, \mathsf{R}(l,(x^\kappa))\}$; for all $\hat{l} \in \mathcal{X}_{\mathbb{L}}^\kappa$ and $\kappa \in \{0,1\}$ with $v(\hat{l}) \in \{0,1\}^2$, $P_{\lambda+1}(\hat{l}) = \mathsf{R}(l,(x^\kappa))$; For all $l \in \mathcal{X}_{\mathcal{L}}^2$, $l = \mathsf{R}(l,(x^2,x^3))$; for all $\hat{l} \in \mathcal{X}_{\mathbb{L}}^2$, $P_{\lambda+1}(\hat{l}) = \mathsf{R}(l,(x^2,x^3))$.*
3. *If $P_1(\hat{l}) = d$ or $v(\hat{l}) = d$ for some $\hat{l} \in \tilde{\mathcal{X}}_{\mathcal{L}} \cup \tilde{\mathcal{X}}_{\mathbb{L}}$ then $P_1(\hat{l}_s) = d$ for all $\hat{l}_s \in \tilde{\mathcal{X}}_{\mathcal{L}}$ and $v(\hat{l}_s) = d$ for all $\hat{l}_s \in \tilde{\mathcal{X}}_{\mathbb{L}}$ which are successors of $\hat{l}$ in the topological order of $\mathcal{H}$.*

4. *There exists some $i \in [n-1]$ such that $P_1(\hat{k}_j) = e$ for all $j \in [i-1]$ and $P_1(\hat{k}_j) = P_1(\hat{k}_{n+1})$ for all $j \in [i:n]$.*

5. *All predicates in (6.18b*) are satisfied.*

6. *For all $i \in [n]$, $\kappa \in \{0,1\}$, for all $u \in V_{T_1}$, $P_2(\hat{x}_i^\kappa) = P_2(\hat{x}_i^{\kappa+2}) = P_1(\hat{k}_{n+1}) = v(u)$; for all $i \in [n]$, $w \in V_{T_2}$ and $\hat{l} \in \tilde{\mathcal{X}}_{\mathcal{L}} \setminus \{\hat{x}_j^\kappa \mid j \in [n], \kappa \in \{0,1\}\}$, $P_2(\hat{k}_i) = P_2(\hat{k}_{n+1}) = P_2(\hat{l}) = v(w)$.*

7. *If $P_2(\hat{k}_{n+1}) = \lambda$ then $P_1(\hat{l}) \in \{0,1\}$ for all $\hat{l} \in \mathcal{X}_{\mathcal{L}}^\lambda$ and $v(\hat{l}_{1,2}) \in \{0,1\}^2$ for all $\hat{l}_{1,2} \in \mathcal{X}_{\mathbb{L}}^\lambda$.*

8. *All predicates in (6.1c*) are satisfied.*

Proof. 1. Follows since all predicates $\mathsf{B}_{\text{copy}(Q_i^\kappa)}$ in (6.13a*) and $\mathsf{B}^\kappa_{\text{copy}_i}$ in (6.13b*) with $i \in [n]$ and $\kappa \in \{0,1\}$ are satisfied by Lemma 6.9.

2. For all $g \in \tilde{\mathcal{G}}$ and $\kappa, \lambda \in \{0,1\}$, predicates $\mathsf{A}_{\text{corr}(g_\lambda)}$ in (6.1a*), respectively $\mathsf{A}_{\text{corr}(g_\lambda)}$ in (6.1a*), $\mathsf{A}_{\text{corr}(g)}$ in (6.1b*) are satisfied because of Lemma 6.9. For all $g \in \mathcal{G}^2$, predicates $\mathsf{B}_{\text{corr}(g_\lambda)}$ in (6.14a*), respectively $\mathsf{B}_{\text{corr}(g_\lambda)}$ in (6.14a*), and $\mathsf{B}_{\text{corr}(g)}$ in (6.14b*), are satisfied because of Lemma 6.9. From this, the claim follows by induction.

3. For all $g \in \tilde{\mathcal{G}}$ and $\lambda \in \{0,1\}$, predicates $\mathsf{A}_{\text{corr}(g_\lambda)}$ in (6.1a*), $\mathsf{A}_{\text{corr}(g_\lambda)}$ in (6.1a*), and $\mathsf{A}_{\text{corr}(g)}$ in (6.1b*) are satisfied because of Lemma 6.9. This implies for every $(1,*)$-gate $g \in \tilde{\mathcal{G}}$ that if $P_1(\hat{l}(g)) = d$ for an input variable, then also $P_1(\hat{l}_s(g)) = d$ for all output variables $\hat{l}_s(g)$ of g. For every $(2,1)$-gate $g \in \tilde{\mathcal{G}}$, all predicates in (6.1a*) and (6.1b*) being satisfied implies that if $P_1(\hat{l}(g)) = d$ for an input variable, then $v(\hat{l}_{1,2}(g)) = d$ and subsequently $P_1(\hat{l}_3(g)) = d$. From this, the claim follows by induction.

4. Assume that such an index i does not exist. This implies that there exists some $j \in [n]$ such that $P_1(\hat{k}_j) \in \{0,1\}$ and $P_1(\hat{k}_{j+1}) \neq P_1(\hat{k}_j)$. In this case, predicate $\mathsf{A}_{\text{carry}(P_j)}$ in (6.7*) is not satisfied; a contradiction to Lemma 6.9.

5. Let $\hat{l}_1(g)$ be an input link variable of a $(1,2)$-gate with two output links $\hat{l}_2(g)$ and $\hat{l}_3(g)$ such that predicate $\mathsf{B}_{\text{copy}(g_\lambda)}$ for some $\lambda \in \{0,1\}$ in (6.18b*) is not satisfied. All predicates in (6.18a*) are satisfied by Lemma 6.9 and therefore $P_2(\hat{l}_2(g)) = P_2(\hat{l}_3(g)) = \mu \in \{0,1\}$; hence, $\mathsf{B}_{\text{copy}(g_{\bar{\lambda}})}$ is also unsatisfied. Setting $P_2(\hat{l}_1(g)) \leftarrow \mu$ improves predicates $\mathsf{B}_{\text{copy}(g_\lambda)}$, $\mathsf{B}_{\text{copy}(g_{\bar{\lambda}})}$ in (6.18b*) and may only violate predicates of smaller weight.

6. By Lemma 6.9, predicates $\mathsf{B}_{\text{join}(Q_i^\kappa)}$ in (6.11b*), $\mathsf{B}_{\text{copy}(P_i)}$ in (6.16*) are satisfied for all $i \in [n]$ and $\kappa \in \{0,1\}$; all predicates for propagation trees T_1 in (6.11a*) and T_2 in (6.18a*) are satisfied; all predicates in (6.18b*) are satisfied by item (5). From this, the claim follows by induction.

7. Suppose that the property does not hold. Recall that in $\mathcal{H}$, constraints are nodes and variables are hyperedges. Let $\hat{l}$ be the smallest link variable with respect to the topological sorting of $\mathcal{H}$ with $v(\hat{l}) = d$ if $\hat{l} \in \mathcal{X}_{\mathbb{L}}^{\bar{\lambda}}$ or $P_1(\hat{l}) = d$ if $\hat{l} \in \mathcal{X}_{\mathcal{L}}^{\bar{\lambda}}$. By item (6), $P_2(\hat{l}') = \lambda$ for all $\hat{l}' \in \mathcal{X}_{\mathcal{L}}^{\lambda}$. For all link variables $\hat{l}_p$ smaller than $\hat{l}$, with respect to the topological sorting of $\mathcal{H}$, $P_1(\hat{l}_p) \neq d$ if $\hat{l}_p \in \mathcal{X}_{\mathcal{L}}^{\lambda}$ or $v(\hat{l}_p) \neq d$ if $\hat{l}_p \in \mathcal{X}_{\mathbb{L}}^{\lambda}$, by definition. The proof now splits on $\hat{l}$:
   a) If $\hat{l} \in \mathcal{X}_{\mathbb{L}}^{\lambda}$ then correctly setting $\hat{l}_{1,2}(g) \in \{0,1\}^2$ does not violate predicates $\mathsf{A}_{\text{corr}(g_\kappa)}$ in (6.1a*) for all $\kappa \in \{0,1\}$ and $\mathsf{A}_{\text{corr}(g)}$ in (6.1b*) and improves $\mathsf{C}^0_{\text{value}(g)}$ in (6.23b*) since $P_2(\hat{l}_3(g)) = \lambda$ as shown above, thus improving the solution.
   b) Now, let $\hat{l} \in \mathcal{X}_{\mathcal{L}}^{\lambda} \setminus \{\hat{x}_1^{\lambda}, \ldots, \hat{x}_n^{\lambda}\}$. Three cases have to be considered:
      i. If $\hat{l} = \hat{z}_i^{\lambda}$ for some $i \in [n]$, then setting $P_1(\hat{z}_i^{\lambda})$ to its correct value in $\{0,1\}$ does not violate the corresponding predicate $\mathsf{A}_{\text{corr}(g)}$ in (6.1a*), $\mathsf{A}^{\kappa}_{\text{join}_i}$ in (6.1c*), and improves $\mathsf{C}_{\text{value}(g)}$ in (6.23c*), thus improving the solution.
      ii. If $\hat{l} = \hat{y}_i^{\lambda}$ for some $i \in [n]$, then setting $P_1(\hat{y}_i^{\lambda})$ to its correct value in $\{0,1\}$ does not violate the corresponding predicate $\mathsf{A}_{\text{corr}(g)}$ in (6.1a*), $\mathsf{B}^{\kappa}_{\text{load}_i}$ in (6.12*) and improves $\mathsf{C}_{\text{value}(g)}$ in (6.23c*), thus improving the solution.
      iii. Otherwise, let $g \in \mathcal{G}^{\lambda}$ be the gate such that $\hat{l}$ is an output variable of g. We distinguish two cases:

         First, let $\hat{l}$ be the input variable to a $(1,*)$-gate $g' \in \mathcal{G}^{\lambda}$. By item (3), $P_1(\hat{l}_o) = d$ for all output links $\hat{l}_o$ of g'. Setting $P_1(\hat{l}_1)$ to its correct binary value does not violate predicates $\mathsf{A}_{\text{corr}(g')}$, $\mathsf{A}_{\text{corr}(g'_\kappa)}$, $\mathsf{A}_{\text{corr}(g)}$, or $\mathsf{A}_{\text{corr}(g_\kappa)}$ in (6.1a*) and $\mathsf{A}_{\text{corr}(g)}$ in (6.1b*) for all $\kappa \in \{0,1\}$ and improves $\mathsf{C}_{\text{value}(g)}$ in (6.23c*), thus improving the solution.

         Now, let $\hat{l}$ be an input variable of a $(2,1)$-gate $g' \in \mathcal{G}^{\lambda}$. By item (3), $v(\hat{l}_{1,2}(g')) = d$. Setting $P_1(\hat{l})$ to its correct binary value does not violate $\mathsf{A}_{\text{corr}(g_\kappa)}$, in (6.1a*) and $\mathsf{A}_{\text{corr}(g)}$ in (6.1b*) and improves $\mathsf{C}_{\text{value}(g)}$ in (6.23b*), thus improving the solution.
8. Let $\lambda = P_2(\hat{k}_{n+1})$. By item (7), $P_1(\hat{z}_i^{\lambda}) \in \{0,1\}$ for all $i \in [n]$. Assume there exists some $i \in [n]$ and $\kappa \in \{0,1\}$ such that predicate $\mathsf{A}^{\kappa}_{\text{join}_i}$ in (6.1c*) is unsatisfied. Setting $\hat{z}_i \leftarrow (P_1(\hat{z}_i^{\lambda}), \lambda)$ does not violate predicate $\mathsf{A}^{\bar{\kappa}}_{\text{join}_i}$ in (6.1c*), improves predicate $\mathsf{A}^{\kappa}_{\text{join}_i}$ in (6.1c*), and may only violate predicates of smaller weight.

□

Lemma 6.10. $P_2(\hat{z}_i) = P_1(\hat{k}_i) = P_2(\hat{k}_{n+1})$ *for all* $i \in [n]$.

Proof. By Corollary 6.2 (1), $P_1(\hat{x}_i^{\kappa}) = x_i^{\kappa+2}$ for all $i \in [n]$ and $\kappa \in \{0,1\}$. By Corollary 6.2 (2), the value computed in best is correct with respect to the input.

By definition of S^2, if best $= \lambda \in \{0,1\}$ then there exists some pos $\in [n]$, such that $Z(x^\lambda)_{\text{pos}} > Z(x^{\bar\lambda})_{\text{pos}}$ and $Z(x^\lambda)_r = Z(x^{\bar\lambda})_r$ for all $r \in [\text{pos} - 1]$; if best $= e$ then for all $r \in [n]$, $Z(x^\lambda)_r = Z(x^{\bar\lambda})_r$ and for simplicity reasons, we will treat this as pos $= n+1$. All predicates in (6.1c*) are satisfied by Corollary 6.2 (8) and therefore $P_1(\hat z_i) \in \{P_1(\hat z_i^0), P_1(\hat z_i^1)\}$ for all $i \in [n]$. Recall that $P_1(\hat z_i) \in \{0,1\}$ by definition.

By Corollary 6.2 (4), there exists some $p \in [n-1]$ such that $P_1(\hat k_j) = e$ for all $j \in [p-1]$ and $P_1(\hat k_j) = P_1(\hat k_{n+1})$ for all $j \in [p:n]$. Let $\lambda = P_2(\hat k_{n+1})$. By Corollary 6.2 (6), $P_2(\hat k_i) = P_2(\hat k_{n+1}) = \lambda$ for all $i \in [n]$.

First, we show that $P_2(\hat z_i) = e$ for all $i \in [p-1]$. Assume that the claim does not hold and let $i \in [p-1]$ be minimal with $P_2(\hat z_i) \neq e$. This implies that predicate $\mathsf{A}_{\text{corr}(D_i)}$ in (6.2*) is unsatisfied. Setting $P_1(\hat k_i) \leftarrow P_2(\hat z_i)$ does not violate predicate $\mathsf{A}_{\text{carry}(P_{i-1})}$ in (6.7*) (only defined if $i > 1$), improves predicate $\mathsf{A}_{\text{corr}(D_i)}$ in (6.2*), and only violates predicates of smaller weight. $P_2(\hat z_{\text{pos}}) \neq e$ since all predicates in (6.1c*) are satisfied by Corollary 6.2 (8). This implies that $p \leq$ pos. In the following, we consider three cases and show that *either $p = 1$ and $P_1(\hat k_i) = P_2(\hat z_i) = P_2(\hat k_{n+1})$ for all $i \in [n]$ or the solution can be improved.*

1. Let $p = 1$ and $P_1(\hat k_1) = \lambda$. Then, Corollary 6.2 (4) implies that $P_1(\hat k_i) = \lambda$ for all $i \in [n+1]$. Assume there exists some $i \in [n]$ such that $P_2(\hat z_i) \neq P_2(\hat k_i)$. The proof now splits on $P_2(\hat z_i)$:
   a) First, let $P_2(\hat z_i) = e$. Setting $P_2(\hat z_i) \leftarrow P_2(\hat k_i)$ does not violate predicates $\mathsf{A}^\kappa_{\text{join}_i}$ in (6.1c*) for all $\kappa \in \{0,1\}$, $\mathsf{A}_{\text{corr}(D_i)}$ in (6.2*), and $\mathsf{A}_{\text{out}(D_i)}$ in (6.6*), improves predicate $\mathsf{C}^0_{\text{tighten}(P_i)}$ in (6.19a*) and only violates predicates of smaller weight.
   b) Now, let $P_2(\hat z_i) = \bar\lambda$. This implies that predicate $\mathsf{A}_{\text{out}(D_i)}$ in (6.6*) is unsatisfied and setting $\hat z_i \leftarrow (P_1(\hat k_i), \lambda)$ improves the solution. If previously, $P_1(\hat z_i) = 0$, then predicate $\mathsf{A}_{\text{corr}(D_i)}$ in (6.2*) improves. Otherwise, predicate $\mathsf{A}_{\text{corr}(D_i)}$ in (6.2*) remains satisfied, predicate $\mathsf{C}^0_{\text{tighten}(P_i)}$ in (6.19a*) improves, and only predicates of smaller weight become violated.
2. Let $p > 1$ and $P_1(\hat k_p) = \lambda$. Then, by definition $P_1(\hat k_{p-1}) = e$ and $P_2(\hat z_{p-1}) = e$ as shown above. Setting $P_1(\hat k_{p-1}) \leftarrow \lambda$ does not violate predicates $\mathsf{A}_{\text{carry}(P_{p-2})}$ in (6.7*) (only defined if $p > 2$), $\mathsf{A}_{\text{corr}(D_{p-1})}$ in (6.2*), $\mathsf{A}_{\text{out}(D_{p-1})}$ in (6.6*), $\mathsf{A}_{\text{carry}(P_{p-1})}$ in (6.7*), $\mathsf{A}_{\text{care}(P_{p-1})}$, and $\mathsf{A}_{\text{care}(P_{p-2})}$ in (6.8*) (only defined in $p > 2$). Predicate $\mathsf{C}^2_{\text{tighten}(P_{p-1})}$ in (6.19d*) improves and only predicate $\mathsf{C}^2_{\text{relax}(P_{p-1})}$ in (6.19e*) of lower weight becomes violated.
3. Let $P_1(\hat k_p) = \bar\lambda$. Recall that $p \in$ [pos]. Predicate $\mathsf{B}_{\text{propagate}}$ in (6.15*) is satisfied by Lemma 6.9 and therefore best $\neq e$; thus, pos $< n+1$. Recall that by definition, $P_1(\hat k_{p-1}) = e$ (only defined if $p > 1$). Two cases have to be considered:
   a) First, let $p <$ pos. Then, $Z(x^0)_p = Z(x^1)_p$ and the proof now splits on the value of $P_2(\hat z_p)$.

i. If $P_2(\hat{z}_p) = e$ then setting $P_1(\hat{k}_p) \leftarrow e$ does not violate predicates $\mathsf{A}_{\mathrm{corr}(D_p)}$ in (6.2⋆), $\mathsf{A}_{\mathrm{carry}(P_{p-1})}$ in (6.7⋆) (only defined if $p > 1$), $\mathsf{A}_{\mathrm{out}(D_p)}$ in (6.6⋆), $\mathsf{A}_{\mathrm{carry}(P_p)}$ in (6.7⋆), and $\mathsf{A}_{\mathrm{care}(P_{p-1})}$ (only defined in $p > 1$), $\mathsf{A}_{\mathrm{care}(P_p)}$ in (6.8⋆). Predicate $\mathsf{C}^2_{\mathrm{tighten}(P_i)}$ in (6.19d⋆) is still unsatisfied but predicate $\mathsf{C}^2_{\mathrm{relax}(P_i)}$ in (6.19e⋆) improves.

ii. Now, let $P_2(\hat{z}_p) \in \{0,1\}$. Setting $P_2(\hat{z}_p) \leftarrow e$ does not violate predicates $\mathsf{A}^{\kappa}_{\mathrm{join}_p}$ in (6.1c⋆) for all $\kappa \in \{0,1\}$, $\mathsf{A}_{\mathrm{corr}(D_p)}$ in (6.2⋆), and $\mathsf{A}_{\mathrm{out}(D_p)}$ in (6.6⋆). Predicate $\mathsf{C}^0_{\mathrm{tighten}(P_p)}$ in (6.19a⋆) remains unsatisfied and predicate $\mathsf{C}^0_{\mathrm{relax}(P_p)}$ in (6.19b⋆) improves.

b) Now, let $p = \mathrm{pos}$. Predicate $\mathsf{A}^{\kappa}_{\mathrm{join}_i}$ in (6.1c⋆) is satisfied for all $\kappa \in \{0,1\}$ by Corollary 6.2 (8) and therefore $P_2(\hat{z}_p) \in \{0,1\}$. This implies that predicate $\mathsf{A}_{\mathrm{corr}(D_p)}$ in (6.2⋆) is satisfied. The proof now splits on the value of $P_2(\hat{z}_p)$:

i. If $P_2(\hat{z}_p) = \lambda$, then setting $P_1(\hat{k}_p) \leftarrow \lambda$ does not violate the heavier predicates $\mathsf{A}_{\mathrm{corr}(D_p)}$ in (6.2⋆) and $\mathsf{A}_{\mathrm{carry}(P_{p-1})}$ in (6.7⋆) (only defined if $p > 1$)), improves predicate $\mathsf{A}_{\mathrm{out}(D_p)}$ in (6.6⋆) and only predicates of lower weight become violated.

ii. If $P_2(\hat{z}_p) = \bar{\lambda}$, then setting $\hat{z}_p \leftarrow (P_1(\hat{z}^{\lambda}_p), \lambda)$ does not violate predicates $\mathsf{A}^{\kappa}_{\mathrm{join}_p}$ in (6.1c⋆) for all $\kappa \in \{0,1\}$, $\mathsf{A}_{\mathrm{corr}(D_p)}$ in (6.2⋆), $\mathsf{A}_{\mathrm{out}(D_p)}$ in (6.6⋆), $\mathsf{C}^0_{\mathrm{tighten}(P_i)}$ in (6.19a⋆), and $\mathsf{C}^0_{\mathrm{relax}(P_i)}$ in (6.19b⋆) and improves predicate $\mathsf{C}^1_{\mathrm{tighten}(P_i)}$ in (6.19c⋆). Note that predicates $\mathsf{A}^{\kappa}_{\mathrm{join}_p}$ in (6.1c⋆) are satisfied for all $\kappa \in \{0,1\}$ and therefore predicate $\mathsf{A}_{\mathrm{out}(D_p)}$ in (6.6⋆) was not satisfied prior to changing the value of $\hat{z}_p$.

□

Lemma 6.11. *Set $\lambda = P_2(\hat{k}_{n+1})$. Then, $P_1(\hat{l}) = d$ for all $\hat{l} \in \mathcal{X}^{\bar{\lambda}}_{\mathcal{L}} \setminus \{\hat{x}^{\bar{\lambda}}_1, \dots, \hat{x}^{\bar{\lambda}}_n\}$, $v(\hat{l}_{1,2}) = d$ for all $\hat{l}_{1,2} \in \mathcal{X}^{\bar{\lambda}}_{\mathbb{L}}$, and $P_1(\hat{x}^{\lambda}_i) = P_1(\hat{y}^{\lambda}_i)$.*

Proof. By Corollary 6.2 (6), $P_2(\hat{k}_{n+1}) = P_2(\hat{l})$ for all $\hat{l} \in \tilde{\mathcal{X}}_{\mathcal{L}} \setminus \{\hat{x}^{\kappa}_1, \dots, \hat{x}^{\kappa}_n\}$ and $P_2(\hat{x}^{\kappa}_i) = P_1(\hat{k}_{n+1})$ for all $i \in [n]$ and $\kappa \in \{0,1\}$. Consider the two claims of the lemma:

1. Recall that in $\mathcal{H}$, constraints are nodes and variables are hyperedges. Let $\hat{l}$ be the largest link variable with respect to the topological sorting of $\mathcal{H}$ with $v(\hat{l}) \neq d$ if $\hat{l} \in \mathcal{X}^{\bar{\lambda}}_{\mathbb{L}}$ or $P_1(\hat{l}) \neq d$ if $\hat{l} \in \mathcal{X}^{\bar{\lambda}}_{\mathcal{L}}$. By Corollary 6.2 (6), $P_2(\hat{l}') = \lambda$ for all $\hat{l}' \in \mathcal{X}^{\bar{\lambda}}_{\mathcal{L}}$. Corollary 6.2 (3) implies that for all link variables $\hat{l}_s$ larger than $\hat{l}$, with respect to the topological sorting of $\mathcal{H}$, $v(\hat{l}_s) = d$ if $\hat{l}_s \in \mathcal{X}^{\bar{\lambda}}_{\mathbb{L}}$ or $P_1(\hat{l}_s) = d$ if $\hat{l}_s \in \mathcal{X}^{\bar{\lambda}}_{\mathcal{L}}$. The proof now splits on $\hat{l}$.

   a) If $\hat{l} \in \mathcal{X}^{\bar{\lambda}}_{\mathbb{L}}$ then setting $v(\hat{l}) \leftarrow d$ does not violate predicates $\mathsf{A}_{\mathrm{corr}(g_\kappa)}$ in (6.1a⋆) for all $\kappa \in \{0,1\}$ and $\mathsf{A}_{\mathrm{corr}(g)}$ in (6.1b⋆) and improves $\mathsf{C}^0_{\mathrm{value}(g)}$ in (6.23b⋆) since $P_2(\hat{l}_3(g)) = \lambda$ as shown above, thus improving the solution.

b) Now, let $\hat{l} \in \mathcal{X}_{\mathcal{L}}^{\bar{\lambda}} \setminus \{\hat{x}_1^{\bar{\lambda}}, \dots, \hat{x}_n^{\bar{\lambda}}\}$. Three cases have to be considered:

   i. If $\hat{l} = \hat{z}_i^{\bar{\lambda}}$ for some $i \in [n]$ then setting $P_1(\hat{z}_i^{\bar{\lambda}}) \leftarrow d$ does not violate the corresponding predicates $\mathsf{A}_{\text{corr}(g)}$ in (6.1a*) and $\mathsf{A}^{\kappa}_{\text{join}_i}$ in (6.1c*) for all $\kappa \in \{0,1\}$, since $P_2(\hat{z}_i) = \lambda$ by Lemma 6.10, improves predicate $\mathsf{C}^1_{\text{value}(g)}$ in (6.23c*) and thus improves the solution.

   ii. If $\hat{l} = \hat{y}_i^{\bar{\lambda}}$ for some $i \in [n]$, then setting $P_1(\hat{y}_i) \leftarrow d$ does not violate the corresponding predicate $\mathsf{A}_{\text{corr}(g)}$ in (6.1a*) and $\mathsf{B}^{\kappa}_{\text{load}_i}$ in (6.12*) for all $\kappa \in \{0,1\}$ since $P_2(\hat{x}_i^{\lambda}) = \lambda$, improves predicate $\mathsf{C}^1_{\text{value}(g)}$ in (6.23c*) and therefore improves the solution.

   iii. Otherwise, let $g \in \mathcal{G}^{\bar{\lambda}}$ be the gate such that $\hat{l}$ is an output variable of g. We distinguish two cases:

   First, let $\hat{l}$ be the input variable to a $(1,*)$-gate $g' \in \mathcal{G}^{\bar{\lambda}}$. Setting $P_1(\hat{l}) \leftarrow d$ does not violate predicates $\mathsf{A}_{\text{corr}(g)}$, $\mathsf{A}_{\text{corr}(g_\kappa)}$, $\mathsf{A}_{\text{corr}(g')}$, or $\mathsf{A}_{\text{corr}(g'_\kappa)}$ in (6.1a*) and $\mathsf{A}_{\text{corr}(g)}$ in (6.1b*) for all $\kappa \in \{0,1\}$, improves $\mathsf{C}^1_{\text{value}(g)}$ in (6.23c*) and thus improves the solution.

   Now, let $\hat{l}$ be an input variable to a $(2,1)$-gate $g' \in \mathcal{G}^{\bar{\lambda}}$. Setting $P_1(\hat{l}) \leftarrow d$ does not violate predicates $\mathsf{A}_{\text{corr}(g)}$ in (6.1a*) and $\mathsf{A}_{\text{corr}(g'_\kappa)}$ in (6.1a*) for all $\kappa \in \{0,1\}$, improves $\mathsf{C}^1_{\text{value}(g)}$ in (6.23c*) and thus improves the solution.

2. Assume there exists some $i \in [n]$ such that $P_1(\hat{x}_i^{\bar{\lambda}}) \neq P_1(\hat{y}_i^{\lambda})$. Since $P_2(\hat{x}_i^{\bar{\lambda}}) = \lambda$, predicate $\mathsf{B}^{\lambda}_{\text{load}_i}$ in (6.12*) is unsatisfied. Setting $P_1(\hat{x}_i^{\bar{\lambda}}) \leftarrow P_1(\hat{y}_i^{\lambda})$ improves predicate $\mathsf{B}^{\lambda}_{\text{load}_i}$ in (6.12*) and does not violate the corresponding heavier predicate $\mathsf{A}_{\text{corr}(g)}$ in (6.1a*), since $P_1(\hat{l}) = d$ for all $\hat{l} \in \mathcal{X}_{\mathcal{L}}^{\bar{\lambda}} \setminus \{\hat{x}_i^{\bar{\lambda}} \mid i \in [n]\}$.

□

Lemma 6.12. *best* $= e$.

Proof. Assume that best $= \lambda \in \{0,1\}$. Then, $c_{\text{C/F}}(x^{\lambda}, I) > c_{\text{C/F}}(x^{\bar{\lambda}}, I)$ by Corollary 6.2, (2) and the definition of circuit S^2. By Lemma 6.11, $P_1(\hat{x}_i^{\bar{\lambda}}) = P_1(\hat{y}_i^{\lambda})$ for all $i \in [n]$. But $c_{\text{C/F}}((P_1(\hat{y}_i^{\lambda}))_{i=1}^n, I) \geq c_{\text{C/F}}(x^{\lambda}, I)$ by definition of variables $\hat{y}_i^{\lambda}$. A contradiction. □

Definition 6.3. *We define a* terminating assignment $\mathcal{B} \in F_{(2,3,6)\text{-MCA}_{2\text{-}par}}(I)$ *to fulfill the following conditions: Set* $\lambda = P_2(\hat{k}_{n+1})$. *For all* $i \in [n]$, $P_1(\hat{k}_i) = P_1(\hat{k}_{n+1}) = \lambda$, *best* $= e$, $P_1(\hat{x}_i^{\bar{\lambda}}) = P_1(\hat{y}_i^{\lambda})$, $P_1(\hat{x}_i^0) = P_1(\hat{x}_i^1) = P_1(\hat{x}_i^2) = P_1(\hat{x}_i^3) = x_i^2 = x_i^3$. *For all* $i \in [n]$, $\kappa \in \{0,1\}$, *and* $u \in V_{T_1}$, $P_2(\hat{x}_i^{\kappa}) = P_2(\hat{x}_i^{\kappa+2}) = P_1(\hat{k}_{n+1}) = v(u)$; *for all* $w \in V_{T_2}$ *and* $\hat{l} \in \tilde{\mathcal{X}}_{\mathcal{L}} \setminus \{\hat{x}_i^{\kappa} \mid i \in [n]\}$, $P_2(\hat{k}_i) = P_2(\hat{k}_{n+1}) = P_2(\hat{l}) = v(w)$. *For all* $\hat{l} \in \mathcal{X}_{\mathcal{L}}^{\lambda}$, $P_1(\hat{l}) = \mathsf{R}(\hat{l}, (x^{\lambda}))$; *for all* $\hat{l} \in \mathcal{X}_{\mathbb{L}}^{\lambda}$ *and* $\kappa \in \{0,1\}$, $P_{\kappa+1}(\hat{l}) = \mathsf{R}(\hat{l}, (x^{\lambda}))$. *For all* $\hat{l}_{1,2} \in \mathcal{X}_{\mathbb{L}}^{\bar{\lambda}}$, $v(\hat{l}_{1,2}) = d$; *for all* $\hat{l} \in \mathcal{X}_{\mathcal{L}}^{\bar{\lambda}} \setminus \{\hat{x}_i^{\bar{\lambda}} \mid i \in [n]\}$, $P_1(\hat{l}) = d$. *For all* $i \in [n]$, $P_1(\hat{z}_i) = P_1(\hat{z}_i^{\lambda}) \in \{0,1\}$ *and* $P_2(\hat{z}_i) = \lambda$. *For all* $l \in \mathcal{X}_{\mathcal{L}}^2$, $l = \mathsf{R}(l, (x^2, x^3))$. *For all* $\hat{l} \in \mathcal{X}_{\mathbb{L}}^2$ *and* $\kappa \in \{0,1\}$, $P_{\kappa+1}(\hat{l}) = \mathsf{R}(l, (x^0, x^1))$.

Lemma 6.13. *Every locally optimal assignment is a terminating assignment.*

Proof. Assume that some assignment A is locally optimal with $\lambda = P_2(\hat{k}_{n+1})$. By Lemma 6.12, best $= e$. By Lemma 6.10, $P_2(\hat{z}_i) = P_1(\hat{k}_i) = P_1(\hat{k}_{n+1}) = \lambda$ for all $i \in [n]$. By Corollary 6.2 (8), all predicates in (6.1c*) are satisfied and this implies that $P_1(\hat{z}_i) = P_1(\hat{z}_i^{\lambda})$. By Lemma 6.11, $P_1(\hat{x}_i^{\bar{\lambda}}) = P_1(\hat{y}_i^{\lambda})$ for all $i \in [n]$. By Corollary 6.2 (2), (7), $P_1(\hat{l}) = \mathsf{R}(\hat{l}, (x^{\lambda}))$ for all $\hat{l} \in \mathcal{X}_{\mathcal{L}}^{\lambda}$, $P_{\kappa+1}(\hat{l}) = \mathsf{R}(\hat{l}, (x^{\lambda}))$ for all $\hat{l} \in \mathcal{X}_{\mathbb{L}}^{\lambda}$ and $\kappa \in \{0,1\}$. By Lemma 6.11, $v(\hat{l}) = d$ for all $\hat{l} \in \mathcal{X}_{\mathbb{L}}^{\bar{\lambda}}$, $P_1(\hat{l}) = d$ for all $\hat{l} \in \mathcal{X}_{\mathcal{L}}^{\bar{\lambda}} \setminus \{\hat{x}_i^{\bar{\lambda}} \mid i \in [n]\}$. By Corollary 6.2 (2), $v(l) = \mathsf{R}(l, (x^2, x^3))$ for all $l \in \mathcal{X}_{\mathcal{L}}^2$ and $P_{\kappa+1}(\hat{l}) = \mathsf{R}(\hat{l}, (x^2, x^3))$ for all $\hat{l} \in \mathcal{X}_{\mathbb{L}}^2$ and $\kappa \in \{0,1\}$. By Corollary 6.2 (6), $P_2(\hat{x}_i^{\kappa}) = P_2(\hat{x}_i^{\kappa+2}) = P_1(\hat{k}_{n+1}) = v(u) = \lambda$ for all $i \in [n]$ $\kappa \in \{0,1\}$, $u \in V_{T_1}$ and $P_2(\hat{k}_i) = P_2(\hat{k}_{n+1}) = P_2(\hat{l}) = v(w) = \lambda$ for all $w \in V_{T_2}$ and $\hat{l} \in \tilde{\mathcal{X}}_{\mathcal{L}} \setminus \{\hat{x}_i^{\kappa} \mid i \in [n], \kappa \in \{0,1\}\}$. Variable best $= e$ and for all $i \in [n]$, $P_1(\hat{x}_i^{\bar{\lambda}}) = P_1(\hat{y}_i^{\lambda})$ by Lemma 6.11 and therefore $P_1(\hat{x}_i^0) = P_1(\hat{x}_i^1)$ by definition of variables $\hat{y}_i^{\kappa}$. $\square$

Definition 6.4 (The Set $\mathscr{R}^\star$)**.** *Fix some solution $\mathsf{a} \in F(\Phi^\star(I))$. For $k \in [n]$, denote $succ(k) := \{l \in \mathcal{L} \mid l$ is on a path in $\mathcal{H}$ from input link X_k to some output link $Z_i, i \in [n]\}$. Let $\mathcal{X}_{succ(k)}^{\lambda}$ denote the set of link variables in $\mathcal{X}_{\mathcal{L}}^{\lambda} \cup \mathcal{X}_{\mathbb{L}}^{\lambda}$ defined by links from $succ(k)$. With slight abuse of notation, we write $\mathsf{a}(\hat{l}) = d$ for $\hat{l} \in \mathcal{X}_{succ(k)}^{\lambda}$ and $k \in [n]$ to denote $\mathsf{a}(P_1(\hat{l})) = d$ if $\hat{l} \in \mathcal{X}_{\mathcal{L}}^{\lambda}$ and $\mathsf{a}(\hat{l}) = d$ if $\hat{l} \in \mathcal{X}_{\mathbb{L}}^{\lambda}$. We define the following predicates, where for $k \in [n]$, $\lambda \in \{0,1\}$*

$$Q_1^\star(\mathsf{a}, \lambda, k) := [\mathsf{a}(\hat{l}) = d \ \forall \ \hat{l} \in \mathcal{X}_{succ(k)}^{\lambda}] \text{ and}$$
$$Q_2^\star(\mathsf{a}, \lambda) := [\mathsf{a}(P_1(\hat{k}_i)) = \mathsf{a}(P_2(\hat{z}_i)) = \lambda \ \forall i \in [n]].$$

We define $\mathscr{R}^\star$ to be the set of all solutions $\mathsf{a}^\star \in F(\Phi^\star(I))$ which satisfy the following properties:

1. *All predicates in* (6.1a*)–(6.1c*) *are satisfied.*
2. *There exists some $\lambda \in \{0,1\}$ such that*
   a) *$\mathsf{a}^\star(x^{\bar{\lambda}}) = \text{Improve}(\mathsf{a}^\star(x^{\lambda}), I)$,*
   b) *$Q_1^\star(\mathsf{a}, \bar{\lambda}, k)$ is satisfied for some $k \in [n]$, and*
   c) *$Q_2^\star(\mathsf{a}, \lambda)$ is satisfied.*

Lemma 6.14. *Let $\lambda \in \{0,1\}$ and let σ be a sequence of improving solutions from $F(\Phi^\star(I))$, where for the first solution $\mathsf{a}_0 \in \sigma$ items* (1), (2b), *and* (2c) *from Definition 6.4 are satisfied and for each solution $\mathsf{a} \in \sigma$, $\mathsf{a}_0(x^{\kappa}) = \mathsf{a}(x^{\kappa})$ for all $\kappa \in \{0,1\}$. Fix some $\mathsf{a} \in \sigma$ and refer to pos(a_0) by pos. Then, all predicates in* (6.1a*)–(6.1c*) *are satisfied in a and*

1. *There exist $p, q \in [n]$ with $p <$ pos and [$q = p$ or $q >$ pos] and*

$$\begin{aligned} \mathsf{a}(P_1(\hat{k}_i)) &= e \ \forall \ i \in [p-1] \\ \mathsf{a}(P_1(\hat{k}_i)) &= \bar{\lambda} \ \forall \ i \in [p : q-1] \\ \mathsf{a}(P_1(\hat{k}_i)) &= \lambda \ \forall \ i \in [q : n]. \end{aligned} \qquad (\text{INV}^\star)$$

2. If $i < pos$ then $\mathsf{a}(P_2(\hat{z}_i)) \in \{e, \mathsf{a}(P_1(\hat{k}_i))\}$.

3. Predicate $\mathsf{A}_{corr(D_i)}$ in (6.2*) is satisfied for all $i \in \{pos\} \cup [q : n]$.

Proof. The proof of this Lemma follows the proof of Lemma 6.7 in Section 6.3. We prove the lemma by induction on the number of improving steps. By definition, (INV*) is satisfied for a_0. Now, let a be some solution in σ. Since a is fixed in the following, we omit the assignment where it is clear from the context, for sake of readability. We show that every step which violates (INV*) does not improve solution a. For this, we fix some $i \in [n]$ and investigate all changes to $P_1(\hat{k}_i)$ and $\hat{z}_i$ which affect (INV*). These variables occur in predicates $\mathsf{A}_{\text{corr}(D_i)}$ in (6.2*), $\mathsf{A}_{\text{out}(D_i)}$ in (6.6*), $\mathsf{A}_{\text{carry}(P_i)}$ in (6.7*), and $\mathsf{A}_{\text{care}(P_i)}$ in (6.8*) for all $i \in [n]$; predicates of smaller weight in which $P_1(\hat{k}_i)$ and $\hat{z}_i$ appear will not be considered. For sake of readability, we again refer to predicates from $\mathsf{A}_{\text{corr}(D_i)}$ in (6.2*) to $\mathsf{A}_{\text{care}(P_i)}$ in (6.8*) only by their number. A predicate with the number (X) but defined by the parameter $i+1$ is denoted by $(X)^+$. By assumption, all predicates in (6.1a*)–(6.1c*) are satisfied in a_0 and as they are the heaviest among the list of predicates, they remain satisfied throughout σ. Hence, we skip these predicates when listing satisfied predicates.

First, consider improvements which originate from setting $P_2(\hat{z}_i)$ with $i \in [n]$ to a new value. We have to show that properties 2 and 3 remain satisfied.

1. Let $i <$ pos and $P_1(\hat{k}_i) = e$. Property 2 is satisfied and therefore $P_2(\hat{z}_i) = e$; hence, predicate (6.2*) is satisfied. Setting $P_2(\hat{z}_i) \leftarrow \beta \in \{0,1\}$ does not improve the solution, since predicate (6.2*) becomes violated.

2. Let $i <$ pos and $P_1(\hat{k}_i) = \mu \in \{0,1\}$. Property 2 is satisfied and therefore $P_2(\hat{z}_i) \in \{e, \mu\}$. Predicate (6.1c*) is satisfied and this implies that $P_1(\hat{z}_i) = z_i^{\mu}$. Consider setting $\hat{z}_i$ to a new value with $P_2(\hat{z}_i) \leftarrow \bar{\mu}$. Predicates in (6.1c*) need to remain satisfied and therefore $\hat{z}_i \leftarrow (z_i^{\bar{\mu}}, \bar{\mu})$. Predicate (6.2*) also needs to remain satisfied and this implies that $z_i^{\bar{\mu}} = 1$. By assumption, $i <$ pos and by definition this implies that $z_i^{\mu} = 1$. This implies that predicate (6.6*) is satisfied in a. Setting $\hat{z}_i \leftarrow (z_i^{\bar{\mu}}, \bar{\mu})$ does not improve the solution since predicate (6.6*) becomes violated.

3. Consider $\hat{z}_i$ for some $i \in \{\text{pos}\} \cup [q : n]$. By property 3, predicate $\mathsf{A}_{\text{corr}(D_i)}$ in (6.2*) is satisfied. Setting $\hat{z}_i$ to a new value in an improving step cannot violate the satisfied predicates $\mathsf{A}^{\kappa}_{\text{join}_i}$ in (6.1c*) for $\kappa \in \{0,1\}$ or $\mathsf{A}_{\text{corr}(D_i)}$ in (6.2*). We will show below that in each improving step, q does not decrease if $q >$ pos. Hence, (INV*) is satisfied after each improving step where the value of $\hat{z}_i$ is changed.

Now, consider variables $P_1(\hat{k}_i)$ for all $i \in [n]$. Recall that by assumption $P_1(\hat{k}_{n-1})$, $P_1(\hat{k}_n) \in \{0,1\}$ and (pos $\in [2 : n-1]$ or pos $= n+1$). We distinguish the following cases.

1. Let $i \leq p-1$. Then, $P_1(\hat{k}_i) = e$ and ($i = 1$ or $P_1(\hat{k}_{i-1}) = e$). Predicates (6.1c*) and (6.2*) are satisfied and therefore $P_2(\hat{z}_i) = e$ and $P_1(\hat{z}_i^0) = P_1(\hat{z}_i^1)$. The proof now splits on $P_1(\hat{k}_{i+1})$:

   a) If $P_1(\hat{k}_{i+1}) = e$ then setting $P_1(\hat{k}_i) \leftarrow \mu \in \{0,1\}$ does not improve the solution, since predicates (6.2*) and (6.6*) do not improve and predicate (6.7*) becomes violated.

   b) If $P_1(\hat{k}_{i+1}) = \mu \in \{0,1\}$ then setting $P_1(\hat{k}_i) \leftarrow \mu$ does not violate (INV*). Setting $P_1(\hat{k}_i) \leftarrow \bar{\mu}$ does not improve the solution, since predicates (6.2*) and (6.6*) do not improve and predicate (6.7*) becomes violated.

2. Let $i = p$ and $p <$ pos. Then $i \leq n-1$, $P_1(\hat{k}_i) = \mu \in \{0,1\}$, ($i = 1$ or $P_1(\hat{k}_{i-1}) = e$), and $P_1(\hat{k}_{i+1}) = \mu$. Property 2 is satisfied and therefore $P_2(\hat{z}_i) \in \{e, \mu\}$. Setting $P_1(\hat{k}_i) \leftarrow \bar{\mu}$ does not improve the solution, since predicates (6.2*) and (6.6*) do not improve and predicate (6.7*) becomes violated. Now, consider setting $P_1(\hat{k}_i) \leftarrow e$. If $P_2(\hat{z}_i) = e$ then setting $P_1(\hat{k}_i) \leftarrow e$ does not violate (INV*). If $P_2(\hat{z}_i) \neq e$ then setting $P_1(\hat{k}_i) \leftarrow e$ does not improve the solution, since predicate (6.2*) becomes violated.

3. Let $i = p =$ pos. Then, $i \leq n-1$, $P_1(\hat{k}_i) = \mu \in \{0,1\}$, ($i = 1$ or $P_1(\hat{k}_{i-1}) = e$) , ($\mu = \bar{\lambda}$ or $\mu = \lambda = P_1(\hat{k}_{i+1})$). Predicates (6.1c*) and (6.2*) are satisfied and therefore $P_2(\hat{z}_i) \neq e$ and ($P_2(\hat{z}_i) = \mu$ or $P_1(\hat{z}_i) = 1$). Setting $P_1(\hat{k}_i) \leftarrow e$ does not improve the solution, since predicate (6.2*) becomes violated. Now, consider setting $P_1(\hat{k}_i) \leftarrow \bar{\mu}$. If $P_2(\hat{z}_i) = \mu$ then setting $P_1(\hat{k}_i) \leftarrow \bar{\mu}$ does not improve the solution, since predicate (6.2*) becomes violated. If $P_2(\hat{z}_i) = \bar{\mu}$, then predicate (6.2*) being satisfied implies that $P_1(\hat{z}_i) = 1$. By definition, $i =$ pos and this implies that $P_2(\hat{z}_i) = \bar{\lambda}$. If $\mu = \lambda$ and $P_2(\hat{z}_i) = \bar{\lambda}$ then setting $P_1(\hat{k}_i) \leftarrow \bar{\mu}$ does not violate (INV*). Otherwise, setting $P_1(\hat{k}_i) \leftarrow \bar{\mu}$ does not improve the solution, since predicate (6.2*) becomes violated.

4. Let $p < i < q$ or $i > q$. Then, $P_1(\hat{k}_{i-1}) = P_1(\hat{k}_i) = \mu \in \{0,1\}$. Setting $P_1(\hat{k}_i) \leftarrow \beta \in \{e, \bar{\mu}\}$ if $i \in [n-2]$ or $P_1(\hat{k}_i) \leftarrow \bar{\mu}$ if $i \in \{n, n-1\}$ does not improve the solution, since predicate (6.7*)$^+$ becomes violated.

   Let us remark that the value of $P_1(\hat{k}_{n+1})$ is unknown. Note that for $i = n$, the above case especially implies that if $P_1(\hat{k}_{n+1}) = \bar{\mu}$ then this cannot lead to a change of $P_1(\hat{k}_n)$ which improves the solution.

5. Let $i = q$ and $q >$ pos. Then, $P_1(\hat{k}_{i-1}) = \bar{\lambda}$, $P_1(\hat{k}_i) = \lambda$, and predicate (6.7*)$^+$ is unsatisfied. Setting $P_1(\hat{k}_i) \leftarrow \bar{\lambda}$ does not violate (INV*). Setting $P_1(\hat{k}_i) \leftarrow e$ if $i \in [n-2]$ does not improve the solution, since predicate (6.7*)$^+$ does not improve and predicate (6.8*)$^+$ becomes violated.

□

Lemma 6.15. $(\Phi^\star, \Psi^\star)$ *is a tight reduction from* Circuit/Flip.

Proof. This proof follows the proof of Lemma 6.8 in Section 6.3. Here, we use the definition of $\mathscr{R}^\star$ given in Definition 6.4. Note that by construction, all terminating assignments are contained in $\mathscr{R}^\star$. Lemma 6.13 implies that $\mathscr{R}^\star$ contains all locally optimal solutions. Now, let $\mathsf{a} \in \mathscr{R}^\star$. We consider an arbitrary sequence σ of improving steps and show for the first solution $\mathsf{a}' \in \mathscr{R}^\star$ with $\mathsf{a}' \neq \mathsf{a}$ which is reached during the sequence, that either $\Psi^\star(I, \mathsf{a}) = \Psi^\star(I, \mathsf{a}')$ or $\Psi^\star(I, \mathsf{a}')$ is a better neighbor of $\Psi^\star(I, \mathsf{a})$. By definition, all predicates in (6.1a*)–(6.1c*) are satisfied in a and as they are the heaviest among the set of predicates, they remain satisfied throughout σ. First, consider the case $c_{\mathrm{C/F}}(\mathsf{a}(x^0), I) = c_{\mathrm{C/F}}(\mathsf{a}(x^1), I)$. Then, $\mathsf{a}(x^0) = \mathsf{a}(x^1)$ since the cost of every solution is unique by assumption. The input vectors $\mathsf{a}(x^0)$ and $\mathsf{a}(x^1)$ cannot be changed in σ, since by (2a), $\mathsf{a}(x^{\bar\lambda}) = \textsc{Improve}(\mathsf{a}(x^\lambda), I)$ and thus $\mathsf{a}(P_1(\hat y_i^\lambda)) \in \{d, \mathsf{a}(P_1(x^{\bar\lambda}))\}$ for all $\lambda \in \{0,1\}$ and $i \in [n]$. This implies that $\Psi^\star(I, \mathsf{a}') = \Psi^\star(I, \mathsf{a}) = \mathsf{a}(x^0)$ for every solution a' in σ.

Now, let $c_{\mathrm{C/F}}(\mathsf{a}(x^{\bar\lambda}), I) > c_{\mathrm{C/F}}(\mathsf{a}(x^\lambda), I)$. We split σ into two phases. Lemma 6.13 implies that in a locally optimal solution $\mathsf{a}^\star \in F(\Phi^\star(I))$, $\mathsf{a}^\star(x^0) = \mathsf{a}^\star(x^1)$. Thus, input vectors x^0 or x^1 have to change at least once.

Phase One: This Phase Continues Until Input Vector x^0 or x^1 Changes Since $Q_1^\star(\mathsf{a}, \bar\lambda, \hat k)$ is satisfied for some $\hat k \in [n]$ and by assumption on $\mathcal{S}$, there exists a path in $\mathcal{S}$ from input link $X_{\hat k}$ to all output links Z_i with $i \in [n]$, therefore $\mathsf{a}(P_1(\hat z_i^{\bar\lambda})) = d$ for all $i \in [n]$. By assumption, all predicates in (6.1c*) are satisfied and therefore $\mathsf{a}(P_1(\hat z_i^\lambda)) \in \{0,1\}$ for all $i \in [n]$. All predicates in (6.1a*) and (6.1b*) are satisfied and therefore $\mathsf{a}(\hat l) \neq d$ for all $\hat l \in \mathcal{X}^\lambda_{\mathrm{succ}(i)}$ and $i \in [n]$. Since $\mathsf{a} \in \mathscr{R}^\star$, $\mathsf{a}(x^{\bar\lambda}) = \textsc{Improve}(\mathsf{a}(x^\lambda), I)$ and there are no predicates which would now become satisfied by changing the value of some $P_1(\hat x_i^{\bar\lambda})$ with $i \in [n]$. For some $P_1(\hat y_i^\lambda)$ to change its binary value in the sequence σ, thus incentivizing changes to $P_1(\hat x_i^{\bar\lambda})$, requires that some $P_1(\hat x_i^\lambda)$ with $i \in [n]$ changes its value to produce the new output. This implies that input vector $\mathsf{a}(x^{\bar\lambda})$ cannot be changed and input vector $\mathsf{a}(x^\lambda)$ changes first. Let $\mathsf{a}^\circ \in F(\Phi^\star(I))$ be a solution which is reached at the end of phase one. There exists some $k \in [n]$ such that $\mathsf{a}^\circ(P_1(\hat y_k^{\bar\lambda})) \in \{0,1\}$, $\mathsf{a}^\circ(P_1(\hat y_k^{\bar\lambda})) \neq \mathsf{a}^\circ(P_1(\hat x_k^\lambda))$ and $\mathsf{a}^\circ(\hat l) = d$ for all $\hat l \in \mathcal{X}^\lambda_{\mathrm{succ}(k)}$. Thus, $Q_1^\star(\mathsf{a}^\circ, \lambda, k)$ is satisfied.

Now, we show that $Q_2^\star(\mathsf{a}^\circ, \bar\lambda)$ is satisfied. Let σ_1 denote the subsequence of σ until the end of phase one. We are under the conditions of Lemma 6.14, when replacing a_0 by a, σ by σ_1, and λ by $\bar\lambda$. Hence, (INV*) holds during σ_1. $Q_1^\star(\mathsf{a}^\circ, \lambda, k)$ is satisfied and this implies that $\mathsf{a}^\circ(P_1(\hat z_i^\lambda)) = d$ for all $i \in [n]$. By Lemma 6.14, all predicates in (6.1c*) are satisfied and this implies that $\mathsf{a}^\circ(P_2(\hat z_i)) = \bar\lambda$ for all $i \in [n]$. Now, we show that $\mathsf{a}^\circ(P_1(\hat k_1)) = \mathsf{a}^\circ(P_1(\hat k_n)) = \bar\lambda$. Lemma 6.14 then implies that $\mathsf{a}^\circ(P_1(\hat k_i)) = \bar\lambda$ for all $i \in [n]$ and hence, $Q_2^\star(\mathsf{a}^\circ, \bar\lambda)$ is satisfied.

1. Recall that we assumed that $Z(\mathsf{x})_1 = 0$ for all $\mathsf{x} \in F_{\mathrm{C/F}}(I)$; hence, $\mathrm{pos}(\mathsf{a}) > 1$. By Lemma 6.14 (2), $\mathsf{a}^\circ(P_1(\hat k_1)) = \mathsf{a}^\circ(P_2(\hat z_1)) = \bar\lambda$.

2. Now, we show that $\mathsf{a}^\circ(P_1(\hat k_n)) = \bar\lambda$. Recall that by definition, $P_1(\hat k_n) \in \{0,1\}$. Assume that $\mathsf{a}^\circ(P_1(\hat k_n)) = \lambda$. Then, by property (3) of (INV*), predicate (6.2*)

is satisfied. This implies that $\mathsf{a}^\circ(P_2(\hat{z}_n)) \in \{e, \lambda\}$; recall that we assumed that $Z(\mathsf{x})_n = 0$ for all $\mathsf{x} \in F_{\mathrm{C/F}}(I)$. A contradiction.

Phase Two: This Phase Starts with Changing the Value of x^λ and Terminates when the Entire Neighbor is Loaded For some $S \subseteq \{x_1^\lambda, \dots, x_n^\lambda\}$, denote $\mathsf{a}(x^\lambda/S)$ the input vector when flipping all input bits $h \in S$. Let h_1 be the bit in which $\mathsf{a}(x^\lambda)$ and $\mathsf{a}(x^{\bar\lambda}) = \textsc{Improve}(\mathsf{a}(x^\lambda), I)$ differ and let h_2 be the bit in which $\mathsf{a}(x^{\bar\lambda})$ and $\textsc{Improve}(\mathsf{a}(x^{\bar\lambda}), I)$ differ. Phase one terminated by reaching a solution a° which satisfies $Q_1^\star(\mathsf{a}^\circ, \lambda, h)$ for some $h \in \{h_1, h_2\}$ and $Q_2^\star(\mathsf{a}, \bar\lambda)$. The input vectors are still unmodified, thus $\mathsf{a}(x^i) = \mathsf{a}^\circ(x^i)$ for all $i \in [0, 3]$. Denote $\mathsf{a}(P_1(\hat{k})) := (\mathsf{a}(P_1(\hat{k}_i)))_{i=1}^n$.

Let setting $P_1(x_h^\lambda) \leftarrow \mathsf{a}^\circ(P_1(y_h^{\bar\lambda}))$ with $h \in \{h_1, h_2\}$, thus satisfying predicate (6.12*) be the first change to $\mathsf{a}^\circ(x^\lambda)$; denote the resulting assignment a^1. By definition of I, $\mathsf{a}(x^\lambda/\{h_1\})$ is the best neighbor of x and this implies that $c_{\mathrm{C/F}}(\mathsf{a}(x^\lambda/\{h_1\}), I) \geq c_{\mathrm{C/F}}(\mathsf{a}(x^\lambda/\{h\}), I)$; hence, $\mathrm{pos}(\mathsf{a}^1) = n + 1$. Let σ' be an arbitrary sequence of improving steps without setting $P_1(x_{\bar h}^\lambda)$ to a new value, where $\bar h \in \{h_1, h_2\} \setminus \{h\}$ and let $\mathsf{a}^\dagger \in \sigma'$ be a solution in σ'. Note that we are under the conditions of Lemma 6.14 when replacing a_0 by a^1, σ by σ', and λ by $\bar\lambda$. Lemma 6.14 now implies that $\mathsf{a}^\dagger(P_1(\hat{k})) = uw$, where $u \in e^*$ and $w \in \bar\lambda^*$. For all $i \in [n]$, predicates $\mathsf{A}_{\mathrm{corr}(D_i)}$ in (6.2*) are satisfied and therefore $\mathsf{a}^\dagger(P_2(\hat{z}_i)) \in \{e, \bar\lambda\}$ and $\mathsf{a}^\dagger(P_1(\hat{z}_i^{\bar\lambda})) \in \{0, 1\}$. All predicates in (6.1a*) and (6.1b*) are satisfied by assumption and this implies that input vector $\mathsf{a}^\dagger(x^{\bar\lambda})$ cannot be modified. No locally optimal solution can be reached this way, and therefore $P_1(x_{\bar h}^\lambda)$ is set to a new value in some solution during σ; denote the resulting solution a'. This requires that $Q_1^\star(\mathsf{a}', \lambda, \bar h)$ is satisfied; with the same argumentation as in phase one, $Q_2^\star(\mathsf{a}', \bar\lambda)$ is satisfied. Hence, we have reached solution $\mathsf{a}' \in \mathscr{R}$ for which $\Psi(I, \mathsf{a}')$ is a better neighbor of $\Psi(I, \mathsf{a})$. □

Theorem 6.2. CIRCUIT/FLIP $\leq_{pls}$ $(2,3,6)$-MCA$_{2\text{-}par}$ *using a tight reduction.*

Proof. By Lemma 6.13, every locally optimal assignment A for $\Phi^\star(I)$ is a final assignment. $\Psi^\star(I, \mathsf{A})$ is locally optimal for I, since $x^0 = x^1$ and this implies that $c_{\mathrm{C/F}}(x^\lambda, I) \geq c_{\mathrm{C/F}}((P_1(\hat{y}_i^{\bar\lambda}))_{i=1}^n, I)$, where $\lambda = \mathrm{best}$. Thus, no improving flip of an input bit is possible for $\Psi^\star(I, \mathsf{A})$. By Lemma 6.15 the reduction is tight.

Now, we show that the resulting set of constraints is bipartite. We slightly extend the construction described up to now in order to allow an easier coloring. First, consider the propagation trees in (6.11a*) and (6.18a*). Each leaf can be colored independent of the colors of the other leaves. This may require to extend the tree by certain nodes of degree two. Next, consider the predicates (6.1a*), (6.1b*), and (6.14a*), (6.14b*), describing the correct work of the circuits. Recall that we assumed that every gate with three links is solely adjacent to gates with two links. Implanting gates with two links into some links allows us to color the predicates (6.1a*), (6.1b*), and (6.14a*), (6.14b*) with two colors independent of the colors given to the inputs and outputs $\hat{x}_i^0$, $\hat{x}_i^1$, $\hat{y}_i^0$, $\hat{y}_i^1$, $\hat{z}_i^0$, $\hat{z}_i^1$, x_i^2, x_i^3 for all $i \in [n]$ and best. We choose the following coloring: For all $i \in [n]$ and $\kappa \in \{0, 1\}$, variables $\hat{x}_i^\kappa$, $x_i^{\kappa+2}$ are blue, all variables $\hat{x}_i^{\kappa+2}$, $\hat{y}_i^\kappa$ are red. For all $i \in [n]$ and $\kappa \in \{0, 1\}$, if i is even then variables

$\hat{z}_i^\kappa$, $\hat{k}_i$ are blue and $\hat{z}_i$ is red, if i is odd then variables $\hat{z}_i^\kappa$, $\hat{k}_i$ are red and $\hat{z}_i$ is blue. Variable $\hat{k}_{n+1}$ has the opposite color of $\hat{k}_n$ and variable best has the color of $\hat{k}_n$. We show that this coloring is a correct 2-coloring by giving the parameter list for each of the remaining constraints. For all $i \in [n]$ and $\kappa \in \{0,1\}$, constraint (6.1c*) has variables $\hat{z}_i^\kappa, \hat{z}_i$, constraint D_i has variables $\hat{z}_i, \hat{k}_i$, constraint P_i has variables $\hat{k}_i, \hat{k}_{i+1}$. For each $i \in [n]$ and $\kappa \in \{0,1\}$, constraint Q_i has variables $\hat{x}_i^{\kappa+2}, \hat{x}_i^\kappa$, each constraint in (6.12*) has variables $\hat{x}_i^{\bar{\kappa}}, \hat{y}_i^\kappa$, each constraint in (6.13b*) has variables $\hat{x}_i^{\kappa+2}, x_i^{\kappa+2}$; constraint (6.15*) has variables $\hat{k}_{n+1}$, best. □

6.5 A Reduction to Binary Logic

In this section, we present a general technique to tightly reduce instances of (p,q,r)-VCA with $p,q \in \mathbb{N}$ and $r \geq 3$ to VCA-instances over solely binary variables.

6.5.1 The Reduction

The approach we present is iterative. We first describe how to reduce the maximum valence of each variable by one in an arbitrary given (p,q,r)-VCA-instance.

Lemma 6.16. *For all $p,q \in \mathbb{N}$ and $r \geq 3$, (p,q,r)-VCA $\leq_{pls}$ $(p,q,r-1)$-VCA using a tight reduction.*

Proof. We present the reduction function Φ and the solution mapping Ψ. Let $I = (\mathscr{C}, \mathscr{X}) \in D_{(p,q,r)\text{-VCA}}$ with $p,q, \in \mathbb{N}$, and $r \geq 3$, variable set $\mathscr{X}$, and constraint set $\mathscr{C}$. We construct an instance $\Phi(I) = (\mathscr{C}', \mathscr{X}') \in D_{(p,q,r-1)\text{-VCA}}$ with a set of variables $\mathscr{X}'$ and a constraint set $\mathscr{C}'$. Recall that we denote the valence of a variable $x \in \mathscr{X}$ by $\mathsf{r}(x)$. Each variable $x \in \mathscr{X}$ with $\mathsf{r}(x) < r$ also belongs to $\mathscr{X}'$. Each variable $x \in \mathscr{X}$ with $\mathsf{r}(x) = r$ is replaced by two variables $x_0, x_1 \in \mathscr{X}'$ where $\mathsf{r}(x_0) = r-1$ and $\mathsf{r}(x_1) = 2$. $\mathscr{X}'$ is the set of variables generated this way. Let Δ be the domain of $\mathscr{X}$, i. e. $\Delta = \prod_{i=1}^n [\mathsf{r}(x_i)]$, where $\mathscr{X}$ has n variables x_i, $i \in [n]$. Let Δ' be the domain of $\mathscr{X}'$ as defined above.

We now define a mapping $h : \Delta' \to \Delta$. If $x \in \mathscr{X}$ with $\mathsf{r}(x) < r$, then also $x \in \mathscr{X}'$ and we set $h(v(x)) = v(x)$. If $x \in \mathscr{X}$ with $\mathsf{r}(x) = r$, then $x_0, x_1 \in \mathscr{X}'$ and we set

$$h(v(x_0), v(x_1)) := \begin{cases} v(x_0) & \text{if } v(x_1) = 0 \\ r & \text{if } v(x_1) = 1. \end{cases}$$

Note that h is not an injective mapping. If $v(x_1) = 1$ then $h(v(x_0), v(x_1)) = r$ for all $v(x_0) \in [r-1]$. In our construction, we need further mappings from Δ' onto Δ. For every $y \in \mathscr{X}$ with $\mathsf{r}(y) = r$, we define a mapping $h_y : \Delta' \to \Delta$. Function h_y differs from h only for the values of $v(y_0), v(y_1)$, where y_0, y_1 are the two variables in $\mathscr{X}'$ associated to $y \in \mathscr{X}$ by our construction and

$$h_y(v(y_0, v(y_1)) := v(y_0).$$

We are now ready to describe the set of constraints $\mathscr{C}'$. Let $\mathsf{a}' \in F_{(p,q,r-1)\text{-VCA}}(\Phi(I))$. Each constraint $C \in \mathscr{C}$ is replaced by a constraint $C' \in \mathscr{C}'$ with parameter list $\{x \mid x \text{ occurs in } C \text{ and } \mathsf{r}(x) \leq r-1\} \cup \{x_0, x_1 \mid x \text{ occurs in } C \text{ and } \mathsf{r}(x) = r\}$ and

$$C'(\mathsf{a}') := C'_0(\mathsf{a}') + \sum\nolimits_{y \in H(C)} C'_y(\mathsf{a}'),$$

where $H(C)$ is the set of variables y with $\mathsf{r}(y) = r$ occuring in the parameter list of C and

$$C'_0(\mathsf{a}') := M \cdot C(h(\mathsf{a}')),$$
$$C'_y(\mathsf{a}') := C(h_y(\mathsf{a}')),$$

where M is a sufficiently large number which will be determined later. The sub-constraints C'_0 for $C \in \mathscr{C}$, are called *high level sub-constraints* and the sub-constraints C'_y for $C \in \mathscr{C}$, where $y \in H(C)$, are called *low level sub-constraints.* An improvement on the high level shall exceed all changes on the low level. This is the case if we choose $M = 1 + m \cdot \mathsf{W}$, where m is the number of constraints in $\mathscr{C}$ and W is the maximum value obtained by any constraint in $\mathscr{C}$ by any setting of the variables. By construction, $\mathsf{r}(x) \leq r-1$ for all $x \in \mathcal{X}'$, the sum of the valences of the variables minus the number of variables in each constraint did not change, and every variable appears in at most q constraints.

Solution Mapping Given a solution $\mathsf{a}' \in F_{(p,q,r-1)\text{-VCA}}(\Phi(I))$, function $\Psi(I, \mathsf{a}')$ returns $\mathsf{a} = h(\mathsf{a}')$. This terminates the description of the reduction. ○

6.5.2 Proving the Correctness and Tightness of the Reduction

We have to show that if a' is a locally optimal solution of $\Phi(I)$ then $\Psi(I, \mathsf{a}')$ is a locally optimal solution of I. We do this by proving the equivalent statement: If $\Psi(I, \mathsf{a}')$ for some $\mathsf{a}' \in D_{(p,q,r-1)\text{-VCA}}$ is not a locally optimal solution of I then a' is not a locally optimal solution of $\Phi(I)$.

So, let $\mathsf{a}' \in D_{(p,q,r-1)\text{-VCA}}$ and assume that $\mathsf{a} = \Psi(I, \mathsf{a}')$ is not locally optimal for I. Hence, there exists some variable $x \in \mathcal{X}$, whose value can be changed such that the solution improves. Let $\mathsf{a}(x) = \alpha$ and let setting $\mathsf{a}(x) \leftarrow \beta$ improve the solution. If $\mathsf{r}(x) < r$ or $\mathsf{r}(x) = r$ and $\alpha \in [0, r-1]$ then the change can be done in $\Phi(I)$ in one step. The solution improves on the high level and this improvement exceeds all changes on the low level by construction. Now, let us assume that $\alpha = r$ and $\beta \in [0, r-1]$. Then $\mathsf{a}'(x_1) = 1$ and $\mathsf{a}'(x_0) = \delta$ for some $\delta \in [0, r-1]$. If $\delta = \beta$, then again setting $\mathsf{a}(x) \leftarrow \beta$ can also be done in $\Phi(I)$ in one step by setting $\mathsf{a}'(x_1) \leftarrow 0$. Also, if $\delta \neq \beta$, but a improves by setting $\mathsf{a}(x) \leftarrow \delta$, then setting $\mathsf{a}'(x_1) \leftarrow 0$ improves a' in $\Phi(I)$. The remaining case is $\alpha = r$, $\beta \in [0, r-1]$, $\mathsf{a}'(x_0) = \delta$, $\mathsf{a}'(x_1) = 1$, and setting $\mathsf{a}(x) \leftarrow \delta$ does not improve the solution a. We will show that in this case, setting $\mathsf{a}'(x_0) \leftarrow \beta$ improves the solution a'.

Let $\mathscr{C}_{\mathsf{x}}$ be the set of constraints in input instance I containing variable x and let $\hat{\mathsf{a}}$ ($\check{\mathsf{a}}$, respectively) be the solution obtained by setting $\mathsf{a}(x) \leftarrow \beta$ ($\mathsf{a}(x) \leftarrow \delta$, respectively).

By assumption, the change from a to $\hat{\mathsf{a}}$ improves the solution and the change from a to $\check{\mathsf{a}}$ does not improve the solution, thus

$$\sum_{C\in\mathscr{C}}[C(\hat{\mathsf{a}})-C(\mathsf{a})]=\sum_{C\in\mathscr{C}_{\mathsf{x}}}[C(\hat{\mathsf{a}})-C(\mathsf{a})]>0,\ \text{and}$$
$$\sum_{C\in\mathscr{C}}[C(\check{\mathsf{a}})-C(\mathsf{a})]=\sum_{C\in\mathscr{C}_{\mathsf{x}}}[C(\check{\mathsf{a}})-C(\mathsf{a})]\le 0.$$

Now, let $\hat{\mathsf{a}}'\in F_{(p,q,r-1)\text{-VCA}}$ be the solution obtained by setting $\mathsf{a}'(x_0)\leftarrow\beta$. Note that $\mathsf{a}'(x_1)=1$ and therefore the change $\mathsf{a}'(x_0)\leftarrow\beta$ does not influence the mapping h and the mappings h_y for $y\neq x$, while $h_x(x_0,x_1)=x_0$, i. e. $h_x(\delta,1)=\delta$ and $h_x(\beta,1)=\beta$. Therefore,

$$\begin{aligned}\sum_{C'\in\mathscr{C}'}[C'(\hat{\mathsf{a}}')-C'(\mathsf{a}')]&=\sum_{C'_x,C\in\mathscr{C}_{\mathsf{x}}}[C'_x(\hat{\mathsf{a}}')-C'_x(\mathsf{a}')]\\&=\sum_{C\in\mathscr{C}_{\mathsf{x}}}[C(\hat{\mathsf{a}})-C(\check{\mathsf{a}})]>0.\end{aligned}$$

Thus, setting $\mathsf{a}'(x_0)\leftarrow\beta$ improves the solution a', as needed.

Considering tightness, we define $\mathscr{R}:=\Delta'$, the set of all solutions of $\Phi(I)$. Fix some solution $\mathsf{a}'\in\mathscr{R}$ and a better neighbor $\tilde{\mathsf{a}}'$ of a' for $\Phi(I)$. By assumption, there exists some variable $x'\in\mathcal{X}'$ whose assignment is flipped between a' and $\tilde{\mathsf{a}}'$. If $\mathsf{a}'(x_1)=\tilde{\mathsf{a}}'(x_1)=1$ and $\mathsf{a}'(x_0)\neq\tilde{\mathsf{a}}'(x_0)$, then $\Psi(I,\tilde{\mathsf{a}}')=\Psi(I,\tilde{\mathsf{a}})$. In all other cases, $\Psi(I,\tilde{\mathsf{a}}')$ is a better neighbor of $\Psi(I,\tilde{\mathsf{a}})$. □

Performing the above construction iteratively and decreasing the maximum valence of the variables in each construction step by 1, we obtain the following result:

Theorem 6.3. *For all $p,q\in\mathbb{N}$ and $r\geq 3$, (p,q,r)-VCA $\leq_{pls}$ $(p,q,2)$-VCA using a tight reduction.*

Proof. Iterating the above described reduction decreases the maximum valence of each r-valued variable $x\in\mathcal{X}$ by one and increases the variable list of each constraint containing x by one auxiliary binary variable. Thus, for each constraint, the sum of the valences of its variables remains constant. By construction, each variable appears in at most q constraints. The correctness of the construction and the tightness of the reduction follows by transitivity of the reduction presented above. Thus, the theorem follows. □

As a direct consequence of Theorem 6.3, we obtain the tight $\mathcal{PLS}$-completeness of $(6,2,2)$-MCA.

Corollary 6.3. Circuit/Flip $\leq_{pls}$ $(6,2,2)$-MCA *using a tight reduction.*

6.6 Conclusion and a Discussion of Question 6

In this chapter, we studied the Maximum Constraint Assignment problem, which is a local search version of the well-known weighted Generalized Maximum Satisfiability problem. Our focus was on the intractability of the subclass (p, q, r)-MCA$_{k\text{-par}}$, where each constraint has length at most p, each variable appears in at most q constraints and each variable takes at most r values; additionally, the set of constraints is k-partite. In order to solely focus on the Maximum Constraint Assignment problem, we first presented an observation in Section 6.1 that all intractability results also extend to the Minimum Constraint Assignment problem with the same restrictions.

Results Obtained After presenting the general method of our reductions in Section 6.2, we first focused on the *tight $\mathcal{PLS}$-completeness* of $(3, 2, 3)$-MCA$_{3\text{-par}}$ in Section 6.3 and $(2, 3, 6)$-MCA$_{2\text{-par}}$ in Section 6.4. Additionally, we presented a general technique to simulate arbitrary (p, q, r)-VCA-instances with VCA-instances over solely binary variables in Subsection 6.5. As a corollary, we obtained that $(6, 2, 2)$-MCA is *tight $\mathcal{PLS}$-complete.* We think that reducing the parameters p and q is important. As we will see in Chapter 7, the tight $\mathcal{PLS}$-completeness of $(3, 2, 3)$-MCA$_{3\text{-par}}$ and especially the fact that each variable appears in at most two constraints will prove crucial in order to demarcate the tractability of SetPacking and SetCover; furthermore, the tight $\mathcal{PLS}$-completeness of $(3, 2, 3)$-MCA$_{3\text{-par}}$ will also play a vital role in order to lower the bounds on the intractability of other local search versions of weighted standard set problems. Besides that, we believe that the results presented in this chapter will prove useful for establishing that further $\mathcal{PLS}$ problems with small parameters are $\mathcal{PLS}$-complete. Overall, this will help sharpen the boundary between $\mathcal{PLS}$-complete and polynomial-time solvable problems.

Discussion of Question 6 The intractability results for $(3, 2, 3)$-MCA$_{3\text{-par}}$ and $(2, 3, 6)$-MCA$_{2\text{-par}}$ are optimal in the sense that $(2, 2, r)$-MCA is polynomial-time solvable for every $r \in \mathbb{N}$. Hence, our results *delimit the tractability* of the Maximum Constraint Assignment problem, *when neglecting the parameter r*. For binary valence, we were able to prove that $(6, 2, 2)$-MCA is intractable, where we are not certain if our result is optimal.

Open Problems Nevertheless, we should mention that our discussion of Question 6 lacks a surrounding answer, as there remain some gaps between the (p, q, r)-MCA problems known to be in $\mathcal{P}$ and the ones we showed $\mathcal{PLS}$-complete in this chapter:

1. The results we presented for $(3, 2, 3)$-MCA$_{3\text{-par}}$ and $(2, 3, 6)$-MCA$_{2\text{-par}}$ are optimal when neglecting the parameter r, but our results do not yield an indication on the tractability of $(3, 2, 2)$-MCA$_{3\text{-par}}$ or $(2, 3, r)$-MCA$_{2\text{-par}}$ for all $r \leq 5$.

2. For binary variables, our intractability result on the one hand only requires that each variable appears in at most two constraints; on the other hand, each constraint is allowed to have length up to six. Our results do not yield an indication on the tractability of $(p, 2, 2)$-MCA for $p \leq 5$.

Hence, the problem of determining the exact bounds of the $\mathcal{PLS}$-completeness of the Maximum Constraint Assignment problem and giving a surrounding answer to Question 6 remains tantalizingly open. Having extended and refined the underlying technique from Krentel [67], we believe that one would probably need substantially new ideas in order to demarcate the tractability of the Maximum Constraint Assignment problem for all three parameters.

Chapter 7

On the Complexity of Local Search for Weighted Standard Set Problems

In this chapter, we show that for most weighted standard set problems introduced in Subsection 2.2.4, computing a locally optimal solution is $\mathcal{PLS}$-complete for the 1-differ neighborhood. This means, that the respective problems are already intractable, when one element describing the solution is allowed to be added, deleted, or exchanged for another element which is not part of the solution. For SetPacking-(k) and SetCover-(k), we delimit the tractability of computing locally optimal solutions for the 2-differ neighborhood. We believe that most $\mathcal{PLS}$-complete problems we investigate in this chapter have the potential to serve as candidates to reduce from in future proofs of intractability.

The remainder of this chapter is organized as follows: In Section 7.1, we present the general technique of our reductions for weighted standard set problems. In Section 7.2, we prove Theorems 7.1 and 7.2; we present the results for each weighted standard set problem in a separate subsection. We summarize our investigation with a conclusion and a discussion of Question 7 in Section 7.3.

7.1 How to Show Intractability of Weighted Standard Set Problems

Before stepping into the reductions in the next section, we outline in Subsection 7.1.1 how the neighborhood structure and the weights occurring in the standard set problems we investigate are related to hardness results. We then present the general technique of our reductions for weighted standard set problems in Subsection 7.1.2.

7.1.1 Neighborhoods, Weights, and Tractability

The hardness of a $\mathcal{PLS}$ problem crucially depends on both the *structure of the neighborhood* and the *weights* occurring in the problem.

If on the one hand, the neighborhood structure limits the options for improvements in every step such that this can be exploited by polynomial-time algorithms, then the problems become easy, *regardless* of the weights. This is the case in SetPacking-(1) and SetCover-(1) where we capitalize on the neighborhood structure with a greedy algorithm. Interestingly, for all other problems we investigate, the neighborhood structure does not trim the complexity; loosely speaking, the neighborhood structure does not interfere with the weights in terms of hardness. For most of the problems, we obtain hardness results for the smallest possible neighborhood of size 1.

If on the other hand, all weights are polynomially bounded then locally optimal solutions can be computed via successive improvements in polynomial time, *regardless* of the neighborhood structure. The problems (h)-CNFSat for some $h \in \mathbb{N}$, $(3,2,3)$-MCA$_{3\text{-par}}$, $(3,2,3)$-MinCA, and MaxCut we reduce from in this chapter were proven to be tight $\mathcal{PLS}$-complete and the weights involved are of *exponential* size. In all our reductions, we preserve the overall structure and range of the weights of the given input instance we reduce from. Usually, we also introduce additional weights which are not part of the input instance. They belong to auxiliary gadgets which are specific to the reduction. The weights involved are either of size one or such that a single weight exceeds the sum of all weights in a given instance.

7.1.2 The General Technique of Our Reductions

For all hardness results given in Section 7.2, we first present the core ideas in a nutshell in the respective subsections before stepping into the corresponding reduction. Similar to many non-trivial reductions in $\mathcal{PLS}$ [4, 27, 31, 95, 98, 104], each reduction (Φ, Ψ) typically consists of two parts: In one part, we encode a given instance I of the input problem in instance $\Phi(I)$ in a rather direct manner, while preserving the overall structure and range of the weights in I. In the other part, which is specific to the reduction and represents a large part of our contribution, we introduce auxiliary gadgets that enforce a particular structure in all locally optimal solutions. Eventually, these gadgets ensure that all locally optimal solutions in $\Phi(I)$ correspond to locally optimal solutions in I. Similar to our constructions, our proofs also consist of two parts:

1. First, we show that all locally optimal solutions in $F(\Phi(I)$ use the gadgets as intended. As a side-effect, this also yields an insight view of the structure of locally optimal solutions in $\Phi(I)$. Depending on the reduction, we call these solutions *standard solutions* or to be *consistent* for some property.

2. Second, we show that all locally optimal solutions in $F(\Phi(I)$ correspond to locally optimal solutions in $F(I)$ under the corresponding reduction (Φ, Ψ). Part 1 now allows to concentrate on the set of all consistent or standard solutions.

Let us stress that reducing from $(3,2,*)$-MCA (resp. $(3,2,*)$-MinCA) is crucial for us to show exact bounds on the tractability of SetPacking (resp. SetCover). In general, we believe that reducing from very restricted but $\mathcal{PLS}$-complete versions of the Maximum Constraint Assignment problem might prove useful for establishing that further $\mathcal{PLS}$ problems with small parameters are $\mathcal{PLS}$-complete.

7.2 The $\mathcal{PLS}$-Complexity of Weighted Standard Set Problems

In this section, we investigate the complexity of computing locally optimal solutions for the weighted standard set problems introduced in Section 2.2.4 and prove Theorems 7.1 and 7.2.

Roadmap of this Section We first present the necessary assumptions and preliminaries for our reductions in Subsection 7.2.1. In Subsections 7.2.2-7.2.10, we study the $\mathcal{PLS}$-complexity of W3DM-(p, q) and X3C-(k) (Subsection 7.2.2), SP-(k) (Subsection 7.2.3), SSp-(k) (Subsection 7.2.4), SC-(k) (Subsection 7.2.5), TS-(k) (Subsection 7.2.6), SB-(k) (Subsection 7.2.7), HS-(k) (Subsection 7.2.8), IP-(k) (Subsection 7.2.9), and CC-(k) (Subsection 7.2.10). The proof of tightness for all our reductions follows a common pattern and therefore, we present a generalized proof of tightness in Subsection 7.2.11. All our intractability results are cumulated in Theorem 7.1, all our tractability results in Theorem 7.2; both theorems are presented in Subsection 7.2.12.

7.2.1 Assumptions and Preliminaries

In this subsection, we present the assumptions and preliminaries for the problems we reduce from. In general, for a given instance I of a $\mathcal{PLS}$ problem we reduce from, let $\mathsf{W} \in \mathbb{N}$ be larger than the sum of all weights occurring in I; details for the corresponding problems are provided below. Recall that MaxCut and (h)-CNFSat are tight $\mathcal{PLS}$-complete for some fixed $h \in \mathbb{N}$ [67, 95, 104], as outlined in Subsections 3.1.3 and 3.1.4. By Theorem 6.1 in Chapter 6, $(3, 2, r)$-MCA and $(3, 2, r)$-MCA$_{3\text{-par}}$ are tight $\mathcal{PLS}$-complete for all $r \geq 3$. By Lemma 6.1 in Chapter 6, all intractability results for (p, q, r)-MCA$_{k\text{-par}}$ extend to (p, q, r)-MinCA$_{k\text{-par}}$ for all $p, q, r, k \in \mathbb{N}$; hence, $(3, 2, r)$-MinCA is also tight $\mathcal{PLS}$-complete.

MaxCut Let $I = (\mathcal{G} = (V, E), \mathsf{w}) \in D_{\text{MC}}$ be a given instance of MaxCut. Without loss of generality, we assume that $V = \{1, \ldots, n\}$ and $\mathcal{G} = (V, E)$ is a clique, i. e. there exists an edge $\{u, v\} \in E$ for each pair of nodes $u, v \in V$ with $u \neq v$. Finally, denote $\mathsf{W} > 1 + \sum_{e \in E} \mathsf{w}(e)$.

Minimum/Maximum Constraint Assignment and (h)-CNFSat Let $I = (\mathcal{C}, \mathcal{X})$ be a given instance of $D_{(3,2,r)\text{-MCA}_{3\text{-par}}}$, $D_{(3,2,r)\text{-MCA}}$ or $D_{(3,2,r)\text{-MinCA}}$ for some $r \in \mathbb{N}$. Without loss of generality, we assume that each constraint $C_i \in \mathcal{C}$ has length 3, every variable $x \in \mathcal{X}$ appears in 2 constraints and takes values from $[r]$. Furthermore, the function value of each constraint $C_i \in \mathcal{C}$ for every assignment is strictly larger than 1. Otherwise, we can ensure the latter property by adding a constant offset of 2 to the function value of each constraint for every assignment. Denote

$$\mathsf{W} > \sum\nolimits_{C_i(x_{i_1},x_{i_2},x_{i_3})\in\mathcal{C}} \sum\nolimits_{\mathsf{a}(x_{i_1}),\mathsf{a}(x_{i_2}),\mathsf{a}(x_{i_3})\in[r]^3} C_i(\mathsf{a}(x_{i_1}),\mathsf{a}(x_{i_2}),\mathsf{a}(x_{i_3})).$$

With slight abuse of notation, we denote the variables occurring in a constraint in I also as *literals*. Let $I' = (\mathcal{C}', \mathcal{X}') \in D_{(h)\text{-CNFSAT}}$ for some $h \in \mathbb{N}$, be a given instance of (h)-CNFSAT. In that case, denote $\mathsf{W} > \sum_{C_i\in\mathcal{C}'} \mathsf{w}_i$. With slight abuse of notation, we also denote the variables occurring in a clause in I' as *literals*; for each variable $x \in \mathcal{X}'$ which is negated in a clause by its constant b, we denote its literal by $\bar{x}$.

7.2.2 On the Complexity of Weighted-3-DimensionalMatching-(p, q) and Exact-Cover-By-3-Sets-(k)

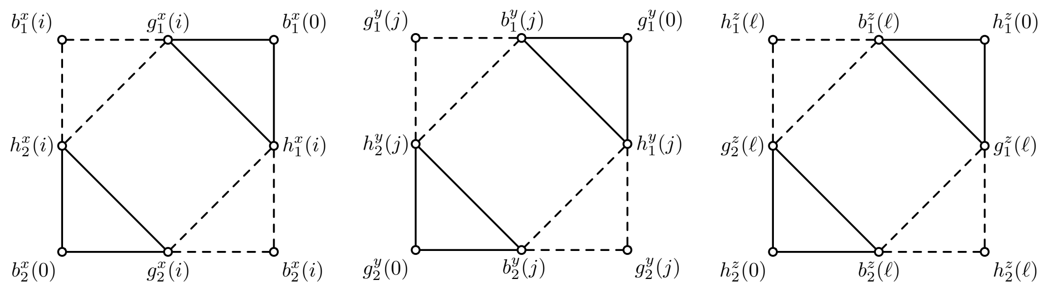
In this subsection, we first show that WEIGHTED-3-DIMENSIONALMATCHING-(p,q) is $\mathcal{PLS}$-complete for all $p \geq 6$ and $q \geq 12$. We present the reduction function Φ and the solution mapping Ψ, which are both slight modifications of a reduction proving that W3DM-$(9,15)$ is $\mathcal{PLS}$-complete, presented in [30]. We also use the notation presented therein. Given an instance $I = (\mathcal{C}, \mathcal{X}) \in D_{(3,2,r)\text{-MCA}_{3\text{-par}}}$, for some $r \in \mathbb{N}$, we construct an instance $\Phi(I) = (N, \mathsf{w}) \in D_{\text{W3DM-}(6,12)}$, consisting of a positive integer $N \in \mathbb{N}$ and a weight function $\mathsf{w} : [N]^3 \to \mathbb{N}_0$ that maps triples to positive integer weights. Recall that for I, we assumed that every constraint has length three and each variable appears in two constraints. Furthermore, the function value of each constraint $C_i \in \mathcal{C}$ for every assignment is strictly larger than 1. By definition, the set of variables $\mathcal{X}$ in instance I is 3-partite, using the colors *blue*, *red*, and *white*. Note that each subset of the set of variables with a certain color has cardinality $|\mathcal{X}|/3$.

Instances of WEIGHTED-3-DIMENSIONALMATCHING-$(6,12)$ constructed by our reduction (Φ, Ψ) can be transformed into instances of EXACT-COVER-BY-3-SETS-(6) such that local optima coincide, by defining each triple as a 3-element set which possesses the weight of the corresponding triple. Therefore, our reduction is also applicable to EXACT-COVER-BY-3-SETS-(6) with the same general construction. This eventually shows that EXACT-COVER-BY-3-SETS-(k) is $\mathcal{PLS}$-complete for all $k \geq 6$.

The Reduction

In a nutshell, the main idea is to *mimic assignments of literals to values in each constraint with triples* which possess the function value of the corresponding constraint for the given assignment. An *additional gadget* ensures that in every locally optimal assignment, all variable assignments induced by the assignment of the respective literals are *consistent*.

In more detail, given instance I, let σ be some order of the variables by their first and second appearance in $\mathcal{C}$. We define $N := 2 \cdot (r \cdot |\mathcal{X}| + |\mathcal{X}|/3)$ and introduce the following weight function w for constructed triples; we categorize the triples by functionality:



(a) Gadget $\mathsf{assign}(i, x)$ for a *blue* variable $x \in \mathcal{X}$ with $i \in [r]$.

(b) Gadget $\mathsf{assign}(j, x)$ for a *red* variable $y \in \mathcal{X}$ with $j \in [r]$

(c) Gadget $\mathsf{assign}(\ell, x)$ for a *white* variable $z \in \mathcal{X}$ with $\ell \in [r]$

Figure 7.1: Gadgets $\mathsf{assign}(i, x)$ for a blue, a red, and a white variable with two large triples (solid triangles) and two medium triples (dashed triangles).

Forcing a Consistent Assignment We define the three sets

$$\begin{aligned}
\mathcal{B} :=& \{b_s^x(i) \mid x \in \mathcal{X}, i \in [r], s \in [2]\} \cup \{b_s^x(0) \mid x \in \mathcal{X} \text{ is a } \textit{blue} \text{ variable}, s \in [2]\},\\
\mathcal{G} :=& \{g_s^x(i) \mid x \in \mathcal{X}, i \in [r], s \in [2]\} \cup \{g_s^x(0) \mid x \in \mathcal{X} \text{ is a } \textit{red} \text{ variable}, s \in [2]\}, \text{and}\\
\mathcal{H} :=& \{h_s^x(i) \mid x \in \mathcal{X}, i \in [r], s \in [2]\} \cup \{h_s^x(0) \mid x \in \mathcal{X} \text{ is a } \textit{white} \text{ variable}, s \in [2]\},
\end{aligned}$$

each of cardinality N. For every *blue* variable $x \in \mathcal{X}$ and $i \in [r]$, we define a gadget $\mathsf{assign}(i, x)$ consisting of two *large triples*

$$(b_1^x(0), g_1^x(i), h_1^x(i)) \text{ and } (b_2^x(0), g_2^x(i), h_2^x(i))$$

of weight $7\mathsf{W}$ and two *medium triples*

$$(b_1^x(i), g_1^x(i), h_2^x(i)) \text{ and } (b_2^x(i), g_2^x(i), h_1^x(i))$$

of weight $2\mathsf{W}$. We depicted a gadget $\mathsf{assign}(i, x)$ in Figure 7.1a for some blue variable $x \in \mathcal{X}$ and $i \in [r]$. For every *red* variable $y \in \mathcal{X}$ and $j \in [r]$, we define a gadget $\mathsf{assign}(j, y)$ consisting of two large triples

$$(b_1^y(j), g_1^y(0), h_1^y(j)) \text{ and } (b_2^y(j), g_2^y(0), h_2^y(j))$$

of weight $7\mathsf{W}$ and two medium triples

$$(b_1^y(j), g_1^y(j), h_2^y(j)) \text{ and } (b_2^y(j), g_2^y(j), h_1^y(j))$$

of weight $2\mathsf{W}$. We again depicted a gadget $\mathsf{assign}(j, y)$ in Figure 7.1b for some red variable $y \in \mathcal{X}$ and $j \in [r]$. For every *white* variable $z \in \mathcal{X}$ and $\ell \in [r]$, we define a gadget $\mathsf{assign}(\ell, z)$ consisting of two large triples

$$(b_1^z(\ell), g_1^z(\ell), h_1^z(0)) \text{ and } (b_2^z(\ell), g_2^z(\ell), h_2^z(0))$$

of weight 7W and two medium triples

$$(b_1^z(\ell), g_2^z(\ell), h_1^z(\ell)) \text{ and } (b_2^z(\ell), g_1^z(\ell), h_2^z(\ell))$$

of weight 2W. We again depicted a gadget $\mathsf{assign}(\ell, z)$ in Figure 7.1c for some white variable $z \in \mathcal{X}$ and $\ell \in [r]$.

Evaluating the Assignment Without loss of generality, let $x \in \mathcal{X}$ be a blue variable, $y \in \mathcal{X}$ be a red variable, and $z \in \mathcal{X}$ be a white variable. For every constraint $C(x, y, z) \in \mathcal{C}$, $i, j, \ell \in [r]$, and $s, t, u \in [2]$, where, with respect to σ, x appears for the s-th, y appears for the t-th time, and z appears for the u-th time, we define *small* triples

$$(b_s^x(i), g_t^y(j), h_u^z(\ell))$$

of weight $C(i, j, \ell)$. All other triples have weight zero. This terminates the description of the reduction function $\Phi(I)$.

Standard Solution Extending the definition from [30], we define a *standard solution* as a solution $\mathsf{S} \in F(\Phi(I))$, consisting of an *assignment part* and an *evaluation part*, of the following form:

1. Considering the assignment part, for every *blue* variable $x \in \mathcal{X}$ there exists some $i \in [r]$, such that for all $s \in [2]$, large triples $(b_s^x(0), g_s^x(i), h_s^x(i)) \in \mathsf{S}$. For all $o \in [r], o \neq i$, medium triples $(b_1^x(o), g_1^x(o), h_2^x(o)), (b_2^x(o), g_2^x(o), h_1^x(o)) \in \mathsf{S}$. For every *red* variable $y \in \mathcal{X}$ there exists some $j \in [r]$, such that for all $s \in [2]$, large triples $(b_s^y(j), g_s^y(0), h_s^y(j)) \in \mathsf{S}$. For all $p \in [r], p \neq j$, medium triples $(b_1^y(p), g_1^y(p), h_2^y(p))$, $(b_2^y(p), g_2^y(p), h_1^y(p)) \in \mathsf{S}$. For every *white* variable $z \in \mathcal{X}$ there exists some $\ell \in [r]$, such that for all $s \in [2]$, large triples $(b_s^z(\ell), g_s^z(\ell), h_s^z(0)) \in \mathsf{S}$. For all $q \in [r], q \neq \ell$, medium triples $(b_1^z(q), g_2^z(q), h_1^z(q))$, $(b_2^z(q), g_1^z(q), h_2^z(q)) \in \mathsf{S}$.
2. Considering the evaluation part, let $x, y, z \in \mathcal{X}$ such that large triples for x, y, and z in S are from gadgets $\mathsf{assign}(i, x)$, $\mathsf{assign}(j, y)$, and $\mathsf{assign}(\ell, z)$ for some $i, j, \ell \in [r]$. Furthermore, let x be a blue variable, y be a red variable, and z be a white variable, without loss of generality. Then, for every constraint $C(x, y, z) \in \mathcal{C}$ and $s, t, u \in [2]$, where x occurs for the s-th, y occurs for the t-th time, and z occurs for the u-th time, with respect to σ, the small triple $(b_s^x(i), g_t^y(j), h_u^z(\ell)) \in \mathsf{S}$.

Solution Mapping Again extending [30], if $\mathsf{S} \in F(\Phi(I))$ is a standard solution, then $\Psi(I, \mathsf{S}) := \mathsf{a}$ with $\mathsf{a} : \mathcal{X} \to [r]$, where for all $x \in \mathcal{X}$

$$\mathsf{a}(x) := \begin{cases} i & \text{if } x \text{ is a } \textit{blue} \text{ variable and } (b_1^x(0), g_1^x(i), h_1^x(i)) \in \mathsf{S} \text{ for some } i \in [r] \\ j & \text{if } x \text{ is a } \textit{red} \text{ variable and } (b_1^x(j), g_1^x(0), h_1^x(j)) \in \mathsf{S} \text{ for some } j \in [r] \\ \ell & \text{if } x \text{ is a } \textit{white} \text{ variable and } (b_1^x(\ell), g_1^x(\ell), h_1^x(0)) \in \mathsf{S} \text{ for some } \ell \in [r]. \end{cases}$$

If $\mathsf{S} \in F(\Phi(I))$ is not a standard solution, then $\Psi(I, \mathsf{S})$ returns the solution a computed by algorithm $\textsc{Init}_{(3,2,r)\text{-MCA}_{3\text{-par}}}(I)$. This terminates the description of the reduction.
○

Lemma 7.1. *Every locally optimal solution $\mathsf{S} \in F(\Phi(I))$ is a standard solution.*

Proof. For sake of completeness, we present the proof of the lemma, as it is similar to the proof of Lemma 1 presented in [30]. Let $\mathsf{S} \in F(\Phi(I))$ be a locally optimal solution. In the following, let $x \in \mathcal{X}$ be a blue variable, without loss of generality.

Roadmap of the Proof of the Lemma

With variable x fixed, the proof splits into three parts. The first two parts show that S contains an assignment part of a standard solution. The third part then shows that S contains an evaluation part of a standard solution. In more detail, we present the following:

1. First, we show that there exist $i, j \in [r]$ such that the two large triples $(b_1^x(0), g_1^x(i), h_1^x(i))$ and $(b_2^x(0), g_2^x(j), h_2^x(j))$ are in S. For every gadget $\mathsf{assign}(*, x)$ without a large triple in S, we prove that there are two medium triples in S.
2. Then, we prove that the two large triples are on the same gadget $\mathsf{assign}(i, x)$ for some $i \in [r]$.
3. Finally, we show that all remaining small triples in S are chosen such that S is a standard solution.

(1): **Two Large Triples and Two Medium Triples** First, consider the large triples. Without loss of generality, assume that for all $i \in [r]$ triple $(b_1^x(0), g_1^x(i), h_1^x(i)) \notin \mathsf{S}$. We construct a better neighboring solution that contains $(b_1^x(0), g_1^x(i), h_1^x(i))$ for some $i \in [r]$. On gadget $\mathsf{assign}(i, x)$, the large triple $(b_1^x(0), g_1^x(i), h_1^x(i))$ of weight $7\mathsf{W}$ is built. The necessary elements $b_1^x(0)$, $g_1^x(i)$, and $h_1^x(i)$ are in at most three triples, each of weight at most $2\mathsf{W}$. Thus, we substitute a total of at most three triples to obtain a strictly better neighboring solution.

Now, consider the medium triples. Assume there exists some $j \in [r]$ such that no large triple and not both medium triples from gadget $\mathsf{assign}(j, x)$ are in S. Without loss of generality, let $(b_1^x(j), g_1^x(j), h_2^x(j)) \notin \mathsf{S}$. On gadget $\mathsf{assign}(j, x)$, the medium triple $(b_1^x(j), g_1^x(j), h_2^x(j))$ of weight $2\mathsf{W}$ is built. The necessary elements are in at most three triples of total weight at most W. Thus, we again substitute a total of three triples to obtain a strictly better neighboring solution.

(2): **Two Large Triples On A Single Gadget** By (1), there are two large triples in S for variable x. Assume that these large triples are placed on two different gadgets $\mathsf{assign}(i, x)$ and $\mathsf{assign}(j, x)$ for some $i, j \in [r]$ with $i \neq j$. In detail, let large triples $(b_2^x(0), g_2^x(i), h_2^x(i)) \in \mathsf{S}$ and $(b_1^x(0), g_1^x(j), h_1^x(j)) \in \mathsf{S}$, without loss of generality. We

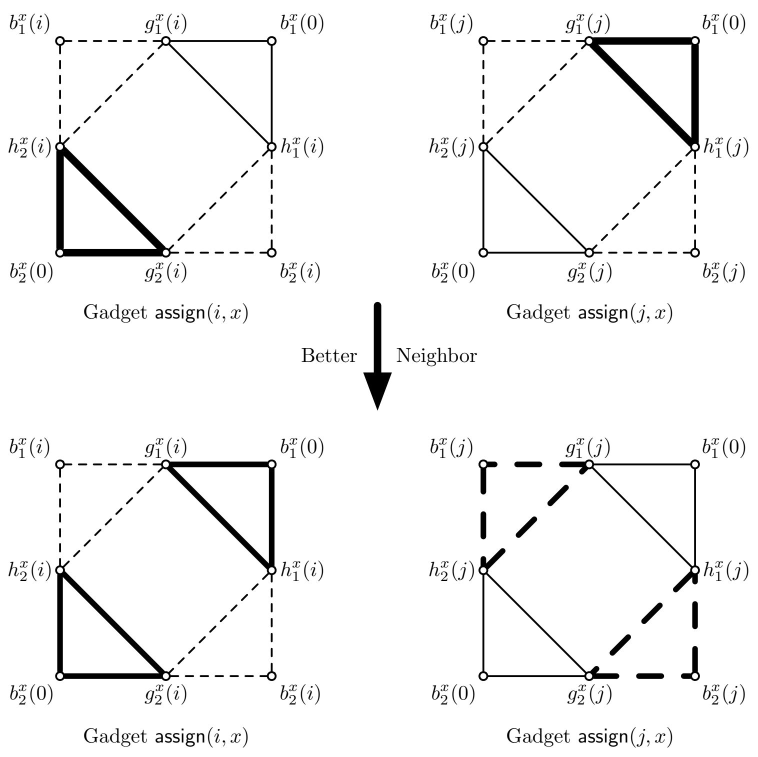


Figure 7.2: Construction of a better neighboring solution, described in (2).

depicted this situation in the upper part of Figure 7.2. Note that by construction, there are no medium triples from gadgets $\mathsf{assign}(i,x)$ or $\mathsf{assign}(j,x)$ in S. We construct a better neighboring solution by removing the large triple $(b_1^x(0), g_1^x(j), h_1^x(j))$ from S. Additionally, on gadget $\mathsf{assign}(i,x)$, the large triple $(b_1^x(0), g_1^x(i), h_1^x(i))$ of weight $7\mathsf{W}$ is built. On gadget $\mathsf{assign}(j,x)$, the two new medium triples $(b_1^x(j), g_1^x(j), h_2^x(j))$ and $(b_2^x(j), g_2^x(j), h_1^x(j))$, each of weight $2\mathsf{W}$, are built. We depicted the better neighboring solution in the lower part of Figure 7.2. Elements $b_1^x(0)$, $g_1^x(j)$, and $h_1^x(j))$ are in the given large triple from gadget $\mathsf{assign}(j,x)$. The remaining elements $g_1^x(i)$, $h_1^x(i)$, $b_1^x(j)$, $h_2^x(j)$, $b_2^x(j)$, and $g_2^x(j)$ are in at most six triples. Our construction does not alter the number of large triples and yields an additional two medium triples, each of weight $2\mathsf{W}$. All triples which are decomposed to obtain the necessary elements for the large triple on gadget $\mathsf{assign}(i,x)$ and for the medium triples on $\mathsf{assign}(j,x)$ have total weight at most W. Thus, we replace a total of at most six triples to obtain a neighboring solution of strictly higher cost.

(3): **Small Weights** The above two parts show that S contains an assignment part of a standard solution. By definition, for every variable $x \in \mathcal{X}$, there exists some $i \in [r]$ such that the two large triples are from the same gadget $\mathsf{assign}(i,x)$. For every blue variable $x \in \mathcal{X}$ this implies that elements $b_1^x(i)$ and $b_2^x(i)$ are not in any large or medium triple in S; analogously for the respective elements for every red and white variable. Without loss of generality, for some $s,t,u \in [2]$ and $i,j,\ell \in [r]$ let $x \in \mathcal{X}$ be a blue variable, $y \in \mathcal{X}$ be a red variable, and $z \in \mathcal{X}$ be a white variable such that $b_s^x(i)$, $g_t^y(j)$, and $h_u^z(\ell)$ are not in any large or medium triple. Let $C(x,y,z) \in \mathcal{C}$ be such that x appears for the s-th time, y appears for the t-th time, and z appears for the u-th time with respect to the given ordering σ. Recall that by assumption, the function value of each constraint $C_i \in \mathcal{C}$ for every assignment is strictly larger than 1. Assume that S deviates in the evaluation part for constraint $C(x,y,z)$. Thus, elements $b_s^x(i)$, $g_t^y(j)$, and $h_u^z(\ell)$ are in at most three triples, each of weight zero. By building the small triple $(b_s^x(i), g_t^y(j), h_u^z(\ell))$ of weight $C(i,j,\ell)$, we replace at most three triples of total weight zero to obtain a neighboring solution with strictly improved cost. $\square$

Lemma 7.2. $(3,2,r)$-MCA$_{3\text{-}par}$ $\leq_{pls}$ W3DM-(p,q) *for all* $p \geq 6$, $q \geq 12$, *and* $r \in \mathbb{N}$.

Proof. Assume there exists a solution $\mathsf{S} \in F(\Phi(I))$ which is locally optimal for $\Phi(I)$, but $\Psi(I,\mathsf{S})$ is not locally optimal for I. By Lemma 7.1, S is a standard solution. Since $\Psi(I,\mathsf{S})$ is not locally optimal for I, there exists a (without loss of generality) white variable $z \in \mathcal{X}$ in instance $I \in D_{(3,2,r)\text{-MCA}_{3\text{-par}}}$, which can be set from value $i \in [r]$ to some value $j \in [r]$ such that the objective function strictly increases by some $\Delta > 0$. Let variable z appear in constraints $C_p, C_q \in \mathcal{C}$. The neighboring solution S' of S, where the two large triples are on gadget $\mathsf{assign}(j,z)$ and two medium triples are on each gadget $\mathsf{assign}(\ell,z)$ for $\ell \in [r]$ with $\ell \neq j$, and all small triples are chosen such that S' is a standard solution improves the cost of S by Δ, by construction. A contradiction to S being locally optimal.

Now, consider the number of boys and girls, which move to new homes. The above described exchange involves the six triples

$$(*,*,h_1^z(0)),(*,*,h_2^z(0)),(*,*,h_1^z(i)),(*,*,h_2^z(i)),(*,*,h_1^z(j)),\text{ and }(*,*,h_2^z(j)).$$

The involved homes are $h_1^z(i)$ and $h_2^z(i)$ from gadget $\mathsf{assign}(i,z)$, homes $h_1^z(j)$ and $h_2^z(j)$ from gadget $\mathsf{assign}(j,z)$ and homes $h_1^z(0)$ and $h_2^z(0)$ which are in each gadget $\mathsf{assign}(*,z)$. On gadget $\mathsf{assign}(i,z)$, girl $g_2^z(i)$ and boy $b_1^z(i)$ move to home $h_1^z(i)$, and girl $g_1^z(i)$ and boy $b_2^z(i)$ move to home $h_2^z(i)$. On gadget $\mathsf{assign}(j,x)$, girl $g_1^z(j)$ and boy $b_1^z(j)$ move to home $h_1^z(0)$, and girl $g_2^z(j)$ and boy $b_2^z(j)$ move to home $h_2^z(0)$. All four boys and girls in small triples move from homes $h_2^z(i)$ and $h_1^z(i)$ to respective homes $h_2^z(j)$ and $h_1^z(j)$. Thus, 12 boys or girls move to new homes. For all red (resp. blue) variables which switch assignment, at most 10 boys or girls move to new homes; note that the smaller number in this case arises from both boys (resp. girls) in small triples remaining in their respective homes. ☐

7.2.3 The Exact Complexity of SetPacking-(k)

In this subsection, we prove that SetPacking-(k) is $\mathcal{PLS}$-complete for all $k \geq 2$ and polynomial-time computable for $k = 1$. Given an instance $I = (\mathcal{C}, \mathcal{X}) \in D_{(3,2,r)\text{-MCA}}$ for some $r \in \mathbb{N}$, we construct an instance $\Phi(I) = (\mathcal{M}, \mathsf{w}, m) \in D_{\text{SP-(2)}}$, consisting of a collection $\mathcal{M}$ from a finite set $\mathcal{B}$, a weight function $\mathsf{w} : \mathcal{M} \to \mathbb{N}_0$ that maps sets in collection $\mathcal{M}$ to positive integer weights, and a positive integer $m \leq |\mathcal{M}|$. Recall that for I we assumed that each constraint $C_i \in \mathcal{C}$ has length 3, every variable $x \in \mathcal{X}$ appears in 2 constraints and takes values from $[r]$. Furthermore, the function value of each constraint $C_i \in \mathcal{C}$ for every assignment is strictly larger than 1.

The Reduction In a nutshell, the main idea is to define *sets which represent assignments of literals to values in each constraint* such that *inconsistent assignments intersect.* The weight of a set corresponds to the function value of the constraint, for the variable assignment the set represents. Additional sets of weight 1 which are *pairwise disjoint* from all other sets we introduce offer a relatively *small incentive* in situations where sets intersect in a solution.

In more detail, given an instance I, let σ be some order of the variables by their first and second appearance in $\mathcal{C}$. We create an instance $\Phi(I)$ of SP-(2) with $m := |\mathcal{C}|$. Sets in collection $\mathcal{M}$ consist of elements from the finite set

$$\mathcal{B} := \{e_i, c_i \mid i \in [m]\} \cup \{x_i \mid x \in \mathcal{X}, i \in [r]\}.$$

Collection $\mathcal{M}$ consists of the following sets: For all $i \in [m]$, we introduce a set $C_i^{\text{SP}} := \{e_i\}$ of weight $\mathsf{w}(C_i^{\text{SP}}) := 1$ in $\mathcal{M}$. For each constraint $C_i(u, v, w) \in \mathcal{C}$ and every assignment $a, b, c \in [r]$, we introduce a set $C_i^{a,b,c}$ of weight $\mathsf{w}(C_i^{a,b,c}) := C_i(a, b, c)$

in $\mathcal{M}$. Here, set $C_i^{a,b,c} := \{c_i\} \cup U_a \cup V_b \cup W_c$, where

$$U_a := \begin{cases} \{u_a\} & \text{if } u \in \mathcal{X} \text{ appears in } C_i(u,v,w) \\ & \text{ for the first time with respect to } \sigma \\ \{u_1, \ldots, u_{a-1}, u_{a+1}, \ldots, u_r\} & \text{otherwise;} \end{cases}$$

analogously for V_b and W_c. We call an element x_j for some variable $x \in \mathcal{X}$ and assignment $j \in [r]$ contained in a set from $\mathcal{M}$ due to the first appearance of x, with respect to σ, *direct representative* of x. For $i \in [m]$, denote

$$\mathcal{I}_i := \{C_i^{a,b,c} \mid C_i^{a,b,c} \in \mathcal{M}, a,b,c \in [r]\}.$$

We say that a collection $\mathcal{I}_i$ or a set $C_i^{a,b,c}$ for some $a,b,c \in [r]$ and $i \in [m]$ is *incident* to a collection $\mathcal{I}_j$ for some $j \in [m]$ if the constraints $C_i, C_j \in \mathcal{C}$ have a common variable.

Solution Mapping We call a solution $\mathsf{S} \in F(\Phi(I))$ *set-consistent* if $|\mathsf{S}| = m$ and for each $i \in [m]$ there is exactly one set $C_i^{a,b,c}$ in S for some $a,b,c \in [r]$, which is pairwise disjoint from all other sets in S. If solution $\mathsf{S} \in F(\Phi(I))$ is set-consistent, then function $\Psi(I,\mathsf{S}) := \mathsf{a}$ with $\mathsf{a} : \mathcal{X} \to [r]$, where for each set $C_i^{*,*,*} \in \mathsf{S}$ and every direct-representative $x_j \in C_i^{*,*,*}$, $\mathsf{a}(x) := j$ for variable $x \in \mathcal{X}$. If S is not set-consistent, then the assignment a computed by $\textsc{Init}_{(3,2,r)\text{-MCA}}(I)$ is returned. This terminates the description of the reduction. ○

Lemma 7.3. *Every locally optimal solution $\mathsf{S} \in F(\Phi(I))$ is set-consistent.*

Proof. First, assume there exists a locally optimal solution $\mathsf{S}' \in F(\Phi(I))$ with $|\mathsf{S}'| < m$. By pigeonhole principle and the construction of our reduction, this implies that there exists a set $C_j^{\text{SP}} \in \mathcal{M}$ with $j \in [m]$ which is not in S'. Adding C_j^{SP} to S' strictly improves the cost of S', since by construction C_j^{SP} is pairwise disjoint from all sets in $\mathcal{M}$. A contradiction to S' being locally optimal.

Now, let $\mathsf{S}' \in F(\Phi(I))$ be a locally optimal solution. We distinguish the following two cases:

1. First, assume there exists a set $C_i^{a,b,c}$ in S' for some $a,b,c \in [r]$ which is not pairwise disjoint from all other sets in S'. Note that this includes the case that at least an additional set $C_i^{d,e,f}$ for some $d,e,f \in [r]$ is in S'; sets $C_i^{a,b,c}, C_i^{d,e,f} \in \mathcal{M}$ are not disjoint, since they share element $c_i \in \mathcal{B}$, by construction. By definition of $\textsc{SetPacking}$-(2), $C_i^{a,b,c}$ does not contribute to the cost of S'. By pigeonhole principle and the construction of our reduction there exists a set $C_j^{\text{SP}} \in \mathcal{M}$ for some $j \in [m]$ which is not in S'. Exchanging set $C_i^{a,b,c} \in \mathsf{S}'$ for set $C_j^{\text{SP}} \in \mathcal{M}$ strictly improves the cost of S', since $\mathsf{w}(C_j^{\text{SP}}) = 1$ and C_j^{SP} is pairwise disjoint from all sets in $\mathcal{M}$. All other sets which did intersect with $C_i^{a,b,c}$ may only become pairwise disjoint and thus improve the cost of S'. A contradiction to S' being locally optimal.

2. Now, assume there exists an $i \in [m]$ such that no set from $\mathcal{I}_i$ is in S'. Let $\mathcal{I}_i$ be incident to collections $\mathcal{I}_o$, $\mathcal{I}_p$, and $\mathcal{I}_q$, with $o, p, q \in [m]$. By item 1, we have that for every $j \in [m]$, at most one set $C_j^{*,*,*} \in \mathsf{S}'$ which is pairwise disjoint from all other sets in S'. If present in S', assume that sets $C_o^{a,*,*}, C_p^{b,*,*}, C_q^{c,*,*} \in \mathsf{S}'$ for some $a, b, c \in [r]$. As shown above, $|\mathsf{S}'| = m$ and by assumption, no set from $\mathcal{I}_i$ is in S'. This implies that there exists a set $C_j^{\text{SP}} \in \mathsf{S}'$ for some $j \in [m]$. Exchanging set $C_j^{\text{SP}} \in \mathsf{S}'$ for set $C_i^{a,b,c} \in \mathcal{M}$—if sets from incident collections are not present in S', choose an arbitrary value for the respective variable—strictly increases the cost of S', since $C_i^{a,b,c}$ is pairwise disjoint from all sets in S' by construction and $\mathsf{w}(C_i^{a,b,c}) > \mathsf{w}(C_j^{\text{SP}})$, by assumption on I. A contradiction to S' being locally optimal.

□

Lemma 7.4. $(3,2,r)$-MCA $\leq_{pls}$ SP-(k) *for all* $r \in \mathbb{N}$ *and* $k \geq 2$.

Proof. Assume there exists a solution $\mathsf{S} \in F(\Phi(I))$ which is locally optimal for $\Phi(I)$, but $\Psi(I, \mathsf{S})$ is not locally optimal for I. By Lemma 7.3, S is set-consistent. Since $\Psi(I, \mathsf{S})$ is not locally optimal for I, there exists a variable $x \in \mathcal{X}$ in instance $I \in D_{(3,2,r)\text{-MCA}}$, which can be set from value $i \in [r]$ to some value $j \in [r]$ such that the objective function strictly increases by some $\Delta > 0$. Let variable x appear in constraints $C_p(x, *, *), C_q(x, *, *) \in \mathcal{C}$. Exchanging the sets $C_p^{i,*,*}$ and $C_q^{i,*,*}$ for sets $C_p^{j,*,*}$ and $C_q^{j,*,*}$ in S yields a set-consistent neighboring solution and by construction this strictly increases the cost of S by Δ. A contradiction to S being locally optimal. □

Despite the intractability result for SetPacking-(k) for all $k \geq 2$, it is possible to compute a locally optimal solution of all instances $I \in D_{\text{SP-(1)}}$ in polynomial time.

Lemma 7.5. SetPacking-(1) *is polynomial-time solvable.*

Proof. Given an instance $I = (\mathcal{M}, \mathsf{w}, m) \in D_{\text{SP-(1)}}$, we use the following algorithm GreedyPacking: Starting from the initial feasible solution $\mathsf{S} := \emptyset$, process all sets in $\mathcal{M}$ by weight in descending order and add the heaviest yet unprocessed set to S, if it is disjoint from all sets $S_i \in \mathsf{S}$ and $|\mathsf{S}| \leq m$ after the addition. In order to prove that each solution computed by GreedyPacking is locally optimal, assume that GreedyPacking terminated and the returned solution $\mathsf{S} \in F_{\text{SP-(1)}}(I)$ is not locally optimal. We distinguish the following three cases, where the cost of S can be strictly improved:

1. Assume there exists a set $S_i \in \mathcal{M}$ with $S_i \notin \mathsf{S}$ which can be added to S. This implies that S_i is pairwise disjoint from all sets in S and $|\mathsf{S}| < m$. Thus, GreedyPacking would have included set S_i. A contradiction.

2. Assume there exists a set $S_j \in \mathsf{S}$ which can be deleted from S. This implies that S_j intersects with some set from S and GreedyPacking would have not included S_j. A contradiction.

3. Assume there exists a set $S_j \in \mathsf{S}$ which can be exchanged for some set $S_\ell \in \mathcal{M}$ with $S_\ell \notin \mathsf{S}$. This implies that S_ℓ is pairwise disjoint from all sets in $\mathsf{S} \setminus \{S_j\}$ and $\mathsf{w}(S_\ell) > \mathsf{w}(S_j)$. Thus, GreedyPacking would have included S_ℓ. A contradiction.

□

7.2.4 On the Complexity of SetSplitting-(k)

In this subsection, we prove that SetSplitting-(k) is $\mathcal{PLS}$-complete for all $k \geq 1$. Given an instance $I = (\mathcal{G} = (V, E), \mathsf{w}_{\mathrm{MC}}) \in D_{\mathrm{MC}}$, we construct an instance $\Phi(I) = (\mathcal{M}, \mathsf{w}) \in D_{\mathrm{SSp}\text{-}(1)}$ consisting of a collection $\mathcal{M}$ from a finite set $\mathcal{B}$ and a weight function $\mathsf{w} : \mathcal{M} \to \mathbb{N}_0$ that maps sets from collection $\mathcal{M}$ to positive integer weights.

The Reduction SetSplitting is also known as Hypergraph-2-Colorability and we use a reduction which does not involve additional gadgets: From instance I, we define an instance $\Phi(I)$ of SSp-(1) with the finite set $\mathcal{B} := V$. For every edge $e = \{u, v\} \in E$, we introduce a set $e^{\mathrm{SSp}} := \{u, v\}$ in $\mathcal{M}$ with $\mathsf{w}(e^{\mathrm{SSp}}) := \mathsf{w}_{\mathrm{MC}}(e)$. Here, for each solution $\mathsf{S} = (S_1, S_2) \in F(\Phi(I))$, function $\Psi(I, \mathsf{S}) := \mathsf{p}$ with $\mathsf{p} : V \to \{0, 1\}$, where $\mathsf{p}(u) := 0$ for all nodes $u \in S_1$ and $\mathsf{p}(v) := 1$ for all nodes $v \in S_2$. This terminates the description of the reduction. ○

Lemma 7.6. MaxCut $\leq_{pls}$ SSp-(k) *for all* $k \geq 1$.

Proof. Assume there exists a solution $\mathsf{S} \in F(\Phi(I))$ which is locally optimal for $\Phi(I)$, but $\Psi(I, \mathsf{S})$ is not locally optimal for I. This implies that there exists a node $v \in V$ in instance $I \in D_{\mathrm{MC}}$ which can switch partition such that now edges $e_{i_1}, \ldots, e_{i_p} \in E$ with $i_1, \ldots, i_p, p \in [|E|]$ are in the cut, edges $e_{j_1}, \ldots, e_{j_q} \in E$ with $j_1, \ldots, j_q, q \in [|E|] \setminus \{i_1, \ldots, i_p\}$ are not in the cut and the cost of $\Psi(I, \mathsf{S})$ strictly increases by $\Delta > 0$. By construction, this implies that in $\Phi(I)$, element $v \in \mathcal{B}$ can switch partition and in the resulting neighboring solution $\mathsf{S}' = (S_1, S_2)$ sets $e_{i_1}^{\mathrm{SSp}}, \ldots, e_{i_p}^{\mathrm{SSp}} \in \mathcal{M}$ are not entirely contained in either S_1 or S_2 and sets $e_{j_1}^{\mathrm{SSp}}, \ldots, e_{j_q}^{\mathrm{SSp}} \in \mathcal{M}$ are entirely contained in either S_1 or S_2. By definition of w, this strictly increases the cost of S by Δ. A contradiction to S being locally optimal. □

7.2.5 The Exact Complexity of SetCover-(k)

In this subsection, we prove that SetCover-(k) is $\mathcal{PLS}$-complete for all $k \geq 2$ and polynomial-time computable for $k = 1$. Given an instance $I = (\mathcal{C}, \mathcal{X}) \in D_{(3,2,r)\text{-MinCA}}$ for some $r \in \mathbb{N}$, we construct an instance $\Phi(I) = (\mathcal{M}, \mathsf{w}) \in D_{\mathrm{SC}\text{-}(2)}$, consisting of a collection $\mathcal{M}$ from a finite set $\mathcal{B}$, and a weight function $\mathsf{w} : \mathcal{M} \to \mathbb{N}_0$ that maps sets in collection $\mathcal{M}$ to positive integer weights.

The Reduction In a nutshell, the main idea is to *reuse the encoding of literal assignments and constraints* as sets, presented for SetPacking-(2) in Subsection 7.2.3, such that for every consistent assignment of literals to values, there exists a covering where no element is covered by two sets of the solution. Shifting the weight of each set by a large constant incentivizes dropping sets which cover elements more than once.

In more detail, given instance I, let σ be some order of the variables by their first and second appearance in $\mathcal{C}$. We create an instance $\Phi(I)$ of SC-(2) with $m := |\mathcal{C}|$ and the finite set

$$\mathcal{B} := \{c_i \mid i \in [m]\} \cup \{x_i \mid x \in \mathcal{X}, i \in [r]\}.$$

For each constraint $C_i(u, v, w) \in \mathcal{C}$ and every assignment $a, b, c \in [r]$, we introduce a set $C_i^{a,b,c}$ of weight

$$\mathsf{w}(C_i^{a,b,c}) := C_i(a, b, c) + \mathsf{W}$$

in $\mathcal{M}$. Here, set $C_i^{a,b,c}$ is defined as in Subsection 7.2.3.

Solution Mapping The definition of an incident set or collection, a set-consistent solution, and the solution mapping $\Psi(I, \mathsf{S})$ is as in Subsection 7.2.3, except that for each solution, which is not set-consistent, the assignment a, with $\mathsf{a} : \mathcal{X} \to [r]$, computed by $\textsc{Init}_{(3,2,r)\text{-}\textsc{MinCA}}(I)$ is now returned. This terminates the description of the reduction. ○

Lemma 7.7. *Every locally optimal solution $\mathsf{S} \in F(\Phi(I))$ is set-consistent.*

Proof. Note that by the construction of our reduction and the definition of $F(\Phi(I))$, $|\mathsf{S}| \geq m$ for each solution $\mathsf{S} \in F(\Phi(I))$; otherwise this would imply that there exists an element $c_i \in \mathcal{B}$ for some $i \in [m]$ which is not contained in the union of all sets in S. This also implies that for all $i \in [m]$ there exists a set $C_i^{a,b,c} \in \mathsf{S}$ for some $a, b, c \in [r]$.

Assume there exists a locally optimal solution $\mathsf{S}' \in F(\Phi(I))$ with $|\mathsf{S}'| > m$. By pigeonhole principle and the construction of our reduction, this is implies that there are two sets $C_i^{a,b,c}$ and $C_i^{d,e,f}$ of total weight at least $2\mathsf{W}$ in S for some $i \in [m]$ and $a, b, c, d, e, f \in [r]$. Note that sets $C_i^{a,b,c}$ and $C_i^{d,e,f}$ are not disjoint. By assumption, S' is a feasible solution and therefore, there exist sets $C_h^{o,*,*}, C_j^{p,*,*}, C_\ell^{q,*,*} \in \mathsf{S}'$ for some $o, p, q \in [r]$ and $h, j, \ell \in [m]$ from collections $\mathcal{I}_h$, $\mathcal{I}_j$, and $\mathcal{I}_\ell$ incident to sets $C_i^{a,b,c}, C_i^{d,e,f} \in \mathsf{S}'$. Exchanging the two sets $C_i^{a,b,c}, C_i^{d,e,f}$ for set $C_i^{o,p,q}$ yields a feasible neighboring solution and strictly decreases the cost of S', since $\mathsf{w}(C_i^{o,p,q}) < 2\mathsf{W}$, by construction. A contradiction to S' being locally optimal.

Assume there exists a locally optimal solution $\mathsf{S}' \in F(\Phi(I))$ with $|\mathsf{S}'| = m$. By pigeonhole principle and the first part of the proof, there exists exactly one $C_i^{a,b,c} \in \mathsf{S}'$ for all $i \in [m]$ and some $a, b, c \in [r]$ which is now pairwise disjoint from all other sets in S'; otherwise, there exists an element $x \in \mathcal{B}$ which is not contained in the union of all sets in S'. □

Lemma 7.8. $(3, 2, r)$-$\textsc{MinCA} \leq_{pls}$ SC-(k) *for all $r \in \mathbb{N}$ and $k \geq 2$.*

Proof. Assume there exists a solution $\mathsf{S} \in F(\Phi(I))$ which is locally optimal for $\Phi(I)$, but $\Psi(I, \mathsf{S})$ is not locally optimal for I. By Lemma 7.7, S is a set-consistent assignment. Since $\Psi(I, \mathsf{S})$ is not locally optimal for I, there exists a variable $x \in \mathcal{X}$ in instance $I \in D_{(3,2,r)\text{-MinCA}}$, which can be set from value $i \in [r]$ to some value $j \in [r]$ such that the objective function strictly decreases by some $\Delta > 0$. Let variable x appear in constraints $C_p(x, *, *), C_q(x, *, *) \in \mathcal{C}$. Exchanging sets $C_p^{i,*,*}$ and $C_q^{i,*,*}$ for sets $C_p^{j,*,*}$ and $C_q^{j,*,*}$ in S yields a feasible and set-consistent neighboring solution and by construction, this strictly decreases the cost of S by Δ. A contradiction to S being locally optimal. □

Despite the intractability result for SetCover-(k) for all $k \geq 2$, it is again possible to compute a locally optimal solution of all instances $I \in D_{\text{SP-(1)}}$ in polynomial time.

Lemma 7.9. SetCover-(1) *is polynomial-time solvable.*

Proof. Given an instance $I = (\mathcal{M}, \mathsf{w}) \in D_{\text{SC-(1)}}$, we use the following algorithm GreedyCover: Starting from the initial feasible solution $\mathsf{S} := \mathcal{M}$, process all sets in S by weight in descending order and remove the heaviest yet unprocessed set, if S is still a legal cover of $\mathcal{B}$ after the removal. In order to prove that each solution computed by GreedyCover is locally optimal, assume that GreedyCover terminated and the returned solution $\mathsf{S} \in F_{\text{SP-(1)}}(I)$ is not locally optimal. We distinguish the following two cases, where the cost of S can be strictly decreased:

1. Assume there exists a set $S_i \in \mathsf{S}$ which can be removed. This implies that S is still a legal cover of $\mathcal{B}$ after the removal of S_i. Thus, GreedyCover would have removed set S_i as well. A contradiction.

2. Assume there exists a set $S_i \in \mathsf{S}$ which can be exchanged for a set $S_j \in \mathcal{M}$ with $S_j \notin \mathsf{S}$ and $\mathsf{w}(S_j) < \mathsf{w}(S_i)$. This implies that set S_j of smaller weight covers all elements $\mathcal{B} \setminus \bigcup_{S_\ell \in (\mathsf{S} \setminus \{S_i\})} S_\ell$, i. e. all elements which are uncovered if S_i would be removed from S. Since S_i has larger weight and $\mathsf{S}' := (\mathsf{S} \setminus \{S_i\}) \cup \{S_j\}$ is still a legal cover, GreedyCover would have deleted S_i from S. A contradiction.

□

A Remark on the Maximum Size of the Sets

In the literature, the two problems SetPacking and SetCover are often considered when each set is of small size. For this, the maximum cardinality of each set in every instance is fixed to a constant $\ell \in \mathbb{N}$. The problems are then referred to as Weighted-ℓ-SetPacking and Weighted-ℓ-SetCover. For Weighted-ℓ-SetPacking, the restriction to sets of size at most ℓ properly includes multi-dimensional matching problems [18]. The Weighted-ℓ-SetCover is of practical relevance, as for example Goldschmidt et al. [42] cite applications in the semiconductor industry and in manufacturing. The quality of approximation algorithms for Weighted-ℓ-SetPacking and Weighted-ℓ-SetCover is measured in dependence on ℓ with an established set of algorithms [18, 47, 53] and inapproximability results [49, 50]. We now take a closer look at the maximum cardinality of each set in our results for SetPacking-(k) and SetCover-(k) for all $k \geq 2$.

In our investigation of SetPacking-(k) and SetCover-(k) in Subsections 7.2.3 and 7.2.5, our primary objective was to show intractability for all $k \geq 2$, regardless of the maximum cardinality of each set. For sake of presentation, we chose not to implement the following improvements directly, but rather present them here: When using the $\mathcal{PLS}$-completeness of $(3,2,3)$-MCA and $(3,2,3)$-MinCA presented in Theorem 6.1 and Lemma 6.1 in Chapter 6 as an advanced starting point, the reductions for SetPacking-(2) and SetCover-(2) both create collections where each set has at most $2+2+2+1=7$ elements. This maximum number of elements is reached in each set that models a constraint, where all three variables appear for the second time. The maximum size of the sets can be further lowered by the following two observations:

1. First, recall that by Theorem 6.1 in Chapter 6, $(3,2,3)$-MCA$_{\text{3-par}}$ is tight $\mathcal{PLS}$-complete. By merging two colors, this implies that the subclass of $(3,2,3)$-MCA where all variables are bi-colored such that no constraint is monochromatic is also $\mathcal{PLS}$-complete. This implies that for a fixed instance $I \in D_{(3,2,3)\text{-MCA}_{\text{3-par}}}$, there exists an ordering of the variables by first and second appearance in the set of constraints such that not all variables appear for the second time in each constraint. Therefore, for each instance $\Phi(I) \in D_{\text{SC-(2)}}$, the maximum number of elements in each set created by the reduction for SetCover-(2) is now at most $2+2+1+1=6$.

2. Additionally, we can conclude from Theorem 6.1 in Chapter 6 that there exists an ordering of the variables by first and second appearance in the set of constraints such that in I not all variables appear for the first time. This can be used in the reduction for SetPacking-(2). Note that for an instance $\Phi(I) \in D_{\text{SP-(2)}}$, all pairwise distinct sets for each $i \in [m]$ also intersect in elements other than c_i, by construction. Hence, element $c_i \in \mathcal{B}$ can be removed from each set $C_i^{*,*,*}$ in $\Phi(I)$. In combination, this lowers the maximum number of elements in each set in $\Phi(I) \in D_{\text{SP-(2)}}$ to $2+2+1=5$.

7.2.6 On the Complexity of TestSet-(k)

In this subsection, we prove that TestSet-(k) is $\mathcal{PLS}$-complete for all $k \geq 1$. Given an instance $I = (\mathcal{G} = (V,E), \mathsf{w}_{\text{MC}}) \in D_{\text{MC}}$, we construct an instance $\Phi(I) = (\mathcal{M}, \mathsf{w}, m) \in D_{\text{TS-(1)}}$. Here, $\Phi(I)$ consists of a collection $\mathcal{M}$ from a finite set $\mathcal{B}$, a weight function $\mathsf{w} : \mathcal{B} \times \mathcal{B} \to \mathbb{N}_0$ that maps tuples of elements of $\mathcal{B}$ to positive integer weights, and a positive integer $m \leq |\mathcal{M}|$. Recall that for I we assumed that $\mathcal{G} = (V,E)$ is a clique, i. e. there exists an edge $\{u,v\} \in E$ for each pair of nodes $u, v \in V$ with $u \neq v$.

The Reduction The main idea consists of two parts: On the one hand, we *encode the assignment of each node to a partition in the choice of the respective singleton sets*, in every locally optimal solution. On the other hand, we *simulate the evaluation of the cut in the weight function* w. Additional *small incentives* reward the *inclusion of singleton sets*, whereas *medium incentives* reward the *inclusion of unique partitions for each node.*

In more detail, given instance I, we construct an instance $\Phi(I)$ of TS-(1) with the finite set $\mathcal{B} := \{v_0, v_1 \mid v \in V\}$. We set $m := |V|$ and define

$$\mathcal{M} := \{\{v_0\}, \{v_1\} \mid v \in V\}.$$

For every edge $e = \{u, v\} \in E$ and $\kappa \in \{0, 1\}$, we define

$$\begin{aligned} \mathsf{w}(u_\kappa, v_{\bar{\kappa}}) &:= \mathsf{w}_{\mathrm{MC}}(e) + \mathsf{W} \\ \mathsf{w}(u_\kappa, v_\kappa) &:= \mathsf{W}. \end{aligned}$$

For each $v \in V$ and $\kappa \in \{0, 1\}$, we define

$$\mathsf{w}(v_\kappa, v_{\bar{\kappa}}) := 1.$$

Solution Mapping We call a solution $\mathsf{S} \in F(\Phi(I))$ *positive-element-consistent* if $|\mathsf{S}| = m$ and for every set $\{v_\kappa\} \in \mathsf{S}$ with $\kappa \in \{0, 1\}$, $\{v_{\bar{\kappa}}\} \notin \mathsf{S}$. If solution $\mathsf{S} \in F(\Phi(I))$ is positive-element-consistent, then function $\Psi(I, \mathsf{S}) := \mathsf{p}$ with $\mathsf{p} : V \to \{0, 1\}$, where $\mathsf{p}(v) := \kappa$ for each $\{v_\kappa\} \in \mathsf{S}$ with $\kappa \in \{0, 1\}$. If S is not positive-element-consistent, then the solution p computed by $\textsc{Init}_{\mathrm{MC}}(I)$ is returned. This terminates the description of the reduction. ○

Lemma 7.10. *Every locally optimal solution $\mathsf{S} \in F(\Phi(I))$ is positive-element-consistent.*

Proof. Assume there exists a locally optimal solution $\mathsf{S}' \in F(\Phi(I))$ with $|\mathsf{S}'| < m$. Since by construction $|\mathcal{M}| = 2m$, there exists a set $\{v_\kappa\} \in \mathcal{M}$ with $\{v_\kappa\} \notin \mathsf{S}'$ for some $\kappa \in \{0, 1\}$. Recall that by definition $|\mathsf{S}'| \geq 1$. Adding $\{v_\kappa\}$ to S' increases the cost of S' by at least 1, since $\mathsf{w}(v_\kappa, u) = \mathsf{w}(u, v_\kappa) \geq 1$ for each $\{u\} \in \mathsf{S}'$, by construction. A contradiction to S' being locally optimal.

Assume there exists a locally optimal solution $\mathsf{S}' \in F(\Phi(I))$ such that S' contains two sets $\{v_0\}, \{v_1\} \in \mathsf{S}'$; recall that by definition, $|\mathsf{S}'| \in [m]$. By pigeonhole principle, there exist sets $\{u_0\}, \{u_1\} \in \mathcal{M}$ with $\{u_0\}, \{u_1\} \notin \mathsf{S}'$. Recall that for I we assumed that $\mathcal{G} = (V, E)$ is a clique, i. e. there exists an edge $\{u, v\} \in E$ for each pair of nodes $u, v \in V$ with $u \neq v$. Exchanging $\{v_0\}$ for $\{u_0\}$ increases the cost of S', since no weight W is lost and the additional weights of $\mathsf{w}(v_1, u_0) = \mathsf{w}(u_0, v_1) \geq \mathsf{W}$ dominate the sum of the weights lost due to the removal of $\{v_0\}$. A contradiction to S' being locally optimal. □

Lemma 7.11. $\textsc{MaxCut} \leq_{pls}$ TS-(k) *for all $k \geq 1$.*

Proof. Assume there exists a solution $\mathsf{S} \in F(\Phi(I))$ which is locally optimal for $\Phi(I)$, but $\Psi(I, \mathsf{S})$ is not locally optimal for I. By Lemma 7.10, S is positive-element-consistent. Since $\Psi(I, \mathsf{S})$ is not locally optimal for I, there exists a node $v \in V$ in instance $I \in D_{\mathrm{MC}}$, which can switch partition such that now edges $e_{i_1}, \ldots, e_{i_p} \in E$ with $i_1, \ldots, i_p, p \in [|E|]$ are in the cut, edges $e_{j_1}, \ldots, e_{j_q} \in E$ with $j_1, \ldots, j_q, q \in [|E|] \setminus \{i_1, \ldots, i_p\}$ are not in the cut and the cost strictly increases by $\Delta > 0$. This implies that in $\Phi(I)$, set $\{v_\kappa\} \in \mathsf{S}$ with $\kappa \in \{0, 1\}$ can be exchanged for set $\{v_{\bar{\kappa}}\} \in \mathcal{M}$. On the one hand, for every node $u \in V$ with $u \neq v$ for which there is an edge $e_\ell = \{u, v\}$ with $e_\ell \in \{e_{i_1}, \ldots, e_{i_p}\}$ we have that set $\{u_\kappa\} \in \mathsf{S}$ and

$$\mathsf{w}(u_\kappa, v_{\bar{\kappa}}) = \mathsf{w}(v_{\bar{\kappa}}, u_\kappa) = \mathsf{w}_{\mathrm{MC}}(e_\ell) + \mathsf{W}.$$

On the other hand, for every node $x \in V$ with $x \neq v$ for which there is an edge $e_t = \{x, v\}$ with $e_t \in \{e_{j_1}, \ldots, e_{j_q}\}$ we have that set $\{x_{\bar{\kappa}}\} \in \mathsf{S}$ and

$$\mathsf{w}(x_{\bar{\kappa}}, v_{\bar{\kappa}}) = \mathsf{w}(v_{\bar{\kappa}}, x_{\bar{\kappa}}) = \mathsf{W},$$

by construction. All other pairs of elements of $\mathcal{B}$ remain unchanged in S. By definition of w, this strictly increases the cost of S by 2Δ. A contradiction to S being locally optimal. ☐

7.2.7 On the Complexity of SetBasis-(k)

In this subsection, we prove that SetBasis-(k) is $\mathcal{PLS}$-complete for all $k \geq 1$. Given an instance $I = (\mathcal{C}, \mathcal{X}) \in D_{(h)\text{-CNFSat}}$ for some $h \in \mathbb{N}$, we construct an instance $\Phi(I) = (\mathcal{M}, \mathsf{w}, m) \in D_{\text{SB-(1)}}$ consisting of a collection $\mathcal{M}$ from a finite set $\mathcal{B}$, a weight function $\mathsf{w} : \mathcal{M} \to \mathbb{N}_0$ that maps sets in collection $\mathcal{M}$ to positive integer weights, and a positive integer $m \leq |\mathcal{M}|$. Let us stress that our reduction is independent of the maximum cardinality $\ell \in \mathbb{N}$ of the sets which may be replaced in an improving step in $\Phi(I)$. Recall that parameter ℓ is not part of the input and plays a crucial role in the definition of SetBasis-(k) as a $\mathcal{PLS}$ problem.

The Reduction In a nutshell, the main idea is to *encode every satisfying assignment of a clause via sets containing the respective literals and possessing the weight of the clause.* This is polynomial in the size of the input since the length of each clause in a given instance $I \in D_{(h)\text{-CNFSat}}$ is at most h. In order for locally optimal solutions to be a collection of singleton sets, we add *large incentives* to include *singleton sets* and *medium incentives* to include *unique literals for each variable.*

In more detail, given instance I, we construct an instance $\Phi(I)$ of SB-(1) with the finite set $\mathcal{B} := \{x, \bar{x} \mid x \in \mathcal{X}\}$ and we define $m := |\mathcal{X}|$. For every $x \in \mathcal{X}$, we introduce two sets

$$C_x^{\text{SB}} := \{x\}, C_{\bar{x}}^{\text{SB}} := \{\bar{x}\}$$

in $\mathcal{M}$ of weight $\mathsf{w}(C_x^{\text{SB}}) = \mathsf{w}(C_{\bar{x}}^{\text{SB}}) := 2\mathsf{W}$. For every $x, y \in \mathcal{X}$ with $x \neq y$, we introduce four sets

$$C_{xy}^{\text{SB}} := \{x, y\}, C_{\bar{x}y}^{\text{SB}} := \{\bar{x}, y\}, C_{x\bar{y}}^{\text{SB}} := \{x, \bar{y}\}, \text{ and } C_{\bar{x}\bar{y}}^{\text{SB}} := \{\bar{x}, \bar{y}\}$$

in $\mathcal{M}$ which all have weight $\mathsf{w}(*) := \mathsf{W}$. We encode each satisfying assignment for every clause $C_i \in \mathcal{C}$ by a set containing the corresponding elements from $\mathcal{B}$, define it to possess the weight of clause C_i, and add it to $\mathcal{M}$. In detail, for every clause $C_i(x_{i_1}, \ldots, x_{i_{h_i}}) \in \mathcal{C}$ of length $h_i \leq h$ and weight w_i and every assignment $\mathsf{a} : \{x_{i_1}, \ldots, x_{i_{h_i}}\} \to \{0,1\}$ which satisfies clause C_i, we introduce sets

$$C_i^{\text{SB}(\mathsf{a})} := \{\varphi(x_{i_1}), \ldots, \varphi(x_{i_{h_i}})\},$$

in $\mathcal{M}$ of weight $\mathsf{w}(C_i^{\text{SB}(\mathsf{a})}) := \mathsf{w}_i$, where for all $x \in \{x_{i_1}, \ldots, x_{i_{h_i}}\}$,

$$\varphi(x) := \begin{cases} x & \text{if } \mathsf{a}(x) = 1 \\ \bar{x} & \text{otherwise.} \end{cases}$$

Solution Mapping We call a solution $\mathsf{S} \in F(\Phi(I))$ *single-set-consistent* if $|S_i| = 1$ for each $S_i \in \mathsf{S}$ and for every $\{x\} \in \mathsf{S}$, $\{\bar{x}\} \notin \mathsf{S}$. Recall that by definition of SetBasis-(k), $|\mathsf{S}| = m$. If solution $\mathsf{S} \in F(\Phi(I))$ is single-set-consistent, then function $\Psi(I, \mathsf{S}) := \mathsf{a}$ with $\mathsf{a} : \mathcal{X} \to \{0,1\}$, where for all $x \in \mathcal{X}$

$$\mathsf{a}(x) := \begin{cases} 1 & \text{if } \{x\} \in \mathsf{S} \\ 0 & \text{otherwise.} \end{cases}$$

If S is not single-set-consistent, then the assignment a computed by $\text{Init}_{(h)\text{-CNFSat}}(I)$ is returned. This terminates the description of the reduction. ○

Lemma 7.12. *Every locally optimal solution $\mathsf{S} \in F(\Phi(I))$ is single-set-consistent.*

Proof. Assume there exists a locally optimal solution $\mathsf{S}' \in F(\Phi(I))$ which contains a set $S_i \in \mathsf{S}'$ with $|S_i| \neq 1$. By definition of SetBasis-(k), $|S_i| > 0$, and therefore $|S_i| \geq 2$. Hence, S_i is only used in the union of sets which by construction have total weight strictly less than $2\mathsf{W}$. By pigeonhole principle, there exists a set $\{x\} \in 2^{\mathcal{B}}$ which is not in S'. Exchanging set S_i for $\{x\}$ strictly increases the cost function. Now, set C_x^{SB} can be constructed and the weight of C_x^{SB} is larger than the sum of the weights of the sets which cannot be constructed any more due to the removal of S_i. A contradiction to S' being locally optimal.

Assume there exists a locally optimal solution $\mathsf{S}' \in F(\Phi(I))$ which contains two sets $\{x\}, \{\bar{x}\} \in \mathsf{S}'$. By pigeonhole principle, there exists a set $\{y\} \in \mathcal{B}$ with $\{y\}, \{\bar{y}\} \notin \mathsf{S}'$. Thus, by exchanging $\{\bar{x}\}$ for $\{y\}$, the additional sets C_{xy}^{SB} and C_{yx}^{SB} can now be constructed. Consider the remaining changes to the cost function by the levels of the weights: Set $C_{\bar{x}}^{\text{SB}}$ cannot be constructed any more and set C_y^{SB} can now be constructed; hence, no weight $2\mathsf{W}$ is lost due to the exchange operation. As shown above, $|S_i| = 1$ for each $S_i \in \mathsf{S}$. For each two sets $C_{\bar{x}z}^{\text{SB}}$ and $C_{z\bar{x}}^{\text{SB}}$ with $\{z\} \in \mathsf{S}'$ and $z \neq x$ which cannot be constructed any more, the two sets C_{yz}^{SB} and C_{zy}^{SB} can now be constructed; hence, no weight W is lost due to the exchange operation. Sets C_{xy}^{SB} and C_{yx}^{SB} both have weight W and this dominates the sum of the weights of the sets which cannot be constructed any more due to the removal of $\{\bar{x}\}$. Thus, the cost of S' strictly increased. A contradiction to S' being locally optimal. □

Lemma 7.13. (h)-CNFSAT $\leq_{pls}$ SB-(k) *for all* $k \geq 1$.

Proof. Assume there exists a solution $\mathsf{S} \in F(\Phi(I))$ which is locally optimal for $\Phi(I)$, but $\Psi(I, \mathsf{S})$ is not locally optimal for I. By Lemma 7.12, S is single-set-consistent. Since $\mathsf{a} = \Psi(I, \mathsf{S})$ is not locally optimal for I, there exists a variable $x \in \mathcal{X}$ in instance $I \in D_{(h)\text{-CNFSAT}}$, which can be flipped such that in the resulting solution $\mathsf{a}' \in F(I)$, clauses $C_{i_1}, \dots, C_{i_p} \in \mathcal{C}$ with $i_1, \dots, i_p, p \in [|\mathcal{C}|]$ become satisfied, clauses $C_{j_1}, \dots, C_{j_q} \in \mathcal{C}$ with $j_1, \dots, j_q, q \in [|\mathcal{C}|] \setminus \{i_1, \dots, i_p\}$ become unsatisfied and the cost strictly increases by $\Delta > 0$. Without loss of generality, let $\{x\} \in \mathsf{S}$. Now, this implies that in $\Phi(I)$, set $\{x\} \in \mathsf{S}$ can be exchanged for set $\{\bar{x}\} \in 2^{\mathcal{B}}$. For some clause $C_i \in \mathcal{C}$, denote by a'_i the assignment of a' restricted to the set of variables in C_i. All sets

$$C_{i_1}^{\text{SB}(\mathsf{a}'_{i_1})}, \dots, C_{i_p}^{\text{SB}(\mathsf{a}'_{i_p})} \in \mathcal{M}$$

can be constructed as the union of a subset of sets of S involving $\{\bar{x}\}$, and all sets

$$C_{j_1}^{\text{SB}(\mathsf{a}'_{j_1})}, \dots, C_{j_q}^{\text{SB}(\mathsf{a}'_{j_q})} \in \mathcal{M}$$

cannot be constructed as the union of a subset of sets of S. Set C_x^{SB} cannot be constructed any more and set $C_{\bar{x}}^{\text{SB}}$ can now be constructed; hence, no weight $2\mathsf{W}$ is lost due to the exchange operation. For each two sets C_{xy}^{SB} and C_{yx}^{SB} with $\{y\} \in \mathsf{S}$ and $y \neq x$ which cannot be composed any more, the two sets $C_{\bar{x}y}^{\text{SB}}$ and $C_{y\bar{x}}^{\text{SB}}$ can now be composed; hence, no weight W is lost due to the exchange operation. These are the only changes. By definition of w, this strictly increases the cost of S by Δ. A contradiction to S being locally optimal. $\square$

7.2.8 On the Complexity of HittingSet-(k)

In this subsection, we prove that HITTINGSET-(k) is $\mathcal{PLS}$-complete for all $k \geq 1$. Given an instance $I = (\mathcal{C}, \mathcal{X}) \in D_{\text{CNFSAT}}$, we construct an instance $\Phi(I) = (\mathcal{M}, \mathsf{w}, m) \in D_{\text{HS-(1)}}$ consisting of a collection $\mathcal{M}$ from a finite set $\mathcal{B}$, a weight function $\mathsf{w} : \mathcal{M} \to \mathbb{N}_0$ mapping sets in collection $\mathcal{M}$ to positive integer weights, and a positive integer $m \leq |\mathcal{B}|$. In this subsection, we say that a set $C \in \mathcal{M}$ is *hit* in solution $\mathsf{S} \in F(\Phi(I))$ if $C \cap \mathsf{S} \neq \emptyset$.

The Reduction In a nutshell, the main idea is to *encode every clause as a set containing the respective literals which possesses the weight of the clause.* In order to ensure consistency of the induced variable assignment in each locally optimal solution, we add *large incentives* to *include at least one literal* from every variable, *but not both.*

In more detail, from instance I, we create an instance $\Phi(I)$ of HS-(1) with the finite set $\mathcal{B} := \{x, \bar{x} \mid x \in \mathcal{X}\}$ and we define $m := |\mathcal{X}|$. For every variable $x \in \mathcal{X}$, we introduce a set

$$C_x^{\text{HS}} := \{x, \bar{x}\}$$

in $\mathcal{M}$ with $\mathsf{w}(C_x^{\text{HS}}) := \mathsf{W}$. For every clause, we introduce a single set possessing the weight of the respective clause which contains the set of variables according to their positive or negative evaluation in the clause. In detail, for every clause $C_i(x_{i_1}, \ldots, x_{i_{h_i}}) \in \mathcal{C}$ of length $h_i \in \mathbb{N}$ and weight w_i, we introduce a set

$$C_i^{\text{HS}} := \{\varphi(x_{i_1}), \ldots, \varphi(x_{i_{h_i}})\}$$

in $\mathcal{M}$ of weight $\mathsf{w}(C_i^{\text{HS}}) := \mathsf{w}_i$, where for all $x_j \in \{x_{i_1}, \ldots, x_{i_{h_i}}\}$,

$$\varphi(x_j) := \begin{cases} x_j & \text{if } b_j = 0 \\ \bar{x}_j & \text{otherwise.} \end{cases}$$

Recall that by definition of CNFSAT, constants b_j take the role of negating variables.

Solution Mapping We call a solution $\mathsf{S} \in F(\Phi(I))$ *element-consistent* if $|\mathsf{S}| = m$ and for every $x \in \mathsf{S}$, $\bar{x} \notin \mathsf{S}$. If solution $\mathsf{S} \in F(\Phi(I))$ is element-consistent, then function $\Psi(I, \mathsf{S}) := \mathsf{a}$ with $\mathsf{a} : \mathcal{X} \to \{0, 1\}$, where for all $x \in \mathcal{X}$

$$\mathsf{a}(x) := \begin{cases} 1 & \text{if } x \in \mathsf{S} \\ 0 & \text{otherwise.} \end{cases}$$

If S is not element-consistent, then the assignment a computed by $\text{INIT}_{\text{CNFSAT}}(I)$ is returned. This terminates the description of the reduction. ○

Lemma 7.14. *Every locally optimal solution $\mathsf{S} \in F(\Phi(I))$ is element-consistent.*

Proof. Assume there exists a locally optimal solution $\mathsf{S}' \in F(\Phi(I))$ with $|\mathsf{S}'| < m$. Since $|\mathcal{B}| = 2m$, there exists an element $x \in \mathcal{B}$ with $x, \bar{x} \notin \mathsf{S}'$. Thus, adding x to S' increases the cost of S', since no weight is lost and set C_x^{HS} with $\mathsf{w}(C_x^{\text{HS}}) = \mathsf{W}$ is hit. A contradiction to S' being locally optimal.

Assume there exists a locally optimal solution $\mathsf{S}' \in F(\Phi(I))$ such that S' contains two elements $x, \bar{x} \in \mathsf{S}'$. By pigeonhole principle, there exists an element $y \in \mathcal{B}$ with $y, \bar{y} \notin \mathsf{S}'$. Exchanging x for y in solution S' increases the cost of S'. All sets of weight W that were previously hit are still hit. Additionally, set C_y^{HS} is now hit and $\mathsf{w}(C_y^{\text{HS}})$ is larger than the sum of all sets $C_i^{\text{HS}} \in \mathcal{M}$ with $C_i(*, \ldots, x, \ldots, *) \in \mathcal{C}$ that where hit due to the membership of $x \in \mathsf{S}'$, by construction. A contradiction to S' being locally optimal. □

Lemma 7.15. CNFSAT $\leq_{pls}$ HS-(k) *for all* $k \geq 1$.

Proof. Assume there exists a solution $\mathsf{S} \in F(\Phi(I))$ which is locally optimal for $\Phi(I)$, but $\Psi(I, \mathsf{S})$ is not locally optimal for I. By Lemma 7.14, S is element-consistent. Since $\Psi(I, \mathsf{S})$ is not locally optimal for I, there exists a variable $x \in \mathcal{X}$ in instance $I \in D_{\text{CNFSAT}}$, which can be flipped such that clauses $C_{i_1}, \ldots, C_{i_p} \in \mathcal{C}$ with $i_1, \ldots, i_p, p \in [|\mathcal{C}|]$ become satisfied, clauses $C_{j_1}, \ldots, C_{j_q} \in \mathcal{C}$ with $j_1, \ldots, j_q, q \in [|\mathcal{C}|] \setminus \{i_1, \ldots, i_p\}$

become unsatisfied and the cost strictly increases by $\Delta > 0$. Without loss of generality, let $x \in \mathsf{S}$. Now, this implies that in $\Phi(I)$, element $x \in \mathsf{S}$ can be exchanged for element $\bar{x} \in \mathcal{B}$ and now sets $C^{\text{HS}}_{i_1}, \dots, C^{\text{HS}}_{i_p} \in \mathcal{M}$ are hit, and sets $C^{\text{HS}}_{j_1}, \dots, C^{\text{HS}}_{j_q} \in \mathcal{M}$ are not hit; set C^{HS}_{x} is still hit and those are the only changes. By definition of w, this strictly increases the cost of S by Δ. A contradiction to S being locally optimal. □

7.2.9 On the Complexity of IntersectionPattern-(k)

In this subsection, we prove that IntersectionPattern-(k) is $\mathcal{PLS}$-complete for all $k \geq 1$. Given an instance $I = (\mathcal{G} = (V, E), \mathsf{w}) \in D_{\text{MC}}$, we construct an instance $\Phi(I) = (A, B, \mathcal{M}) \in D_{\text{IP-(1)}}$ consisting of two symmetric $n \times n$ matrices $A = (a_{ij})_{i,j \in [n]}$, and $B = (b_{ij})_{i,j \in [n]}$, and a collection $\mathcal{M}$ from a finite set $\mathcal{B}$. Recall that for I we assumed that $V = \{1, \dots, n\}$ and $\mathcal{G} = (V, E)$ is a clique, i. e. there exists an edge $\{u, v\} \in E$ for each pair of nodes $u, v \in V$ with $u \neq v$. In this reduction, we also assume that for each pair of nodes $u, v \in V$ with $u \neq v$, the lexicographical order agrees with the labeling of the nodes, i. e. $u < v$. In the following, denote $\gamma := |V| - 1$, the number of incident nodes of each node $v \in V$.

The Reduction In a nutshell, the main idea is for each node $v \in V$ and each assignment of v to a partition to introduce *sets of identical cardinality* which have *distinct* cardinality from all other sets. A set representing a certain assignment of a partition to v contains *elements which encode cuts for all edges where node v is an endpoint of.* If an edge $e = \{u, v\} \in E$ is in the cut by a given partition, then the intersection of the two corresponding sets for u and v has cardinality two. In this case, the weight of the edge is added to the solution. Large incentives ensure that—identified by cardinality—the sets for variables are placed in the right position in every locally optimal solution.

In more detail, given instance I, define $n := |V|$. We create an instance $\Phi(I)$ of IP-(1) with the finite set

$$\begin{aligned}\mathcal{B} := \{&v^e_\kappa \mid v \in V, \kappa \in \{0,1\}, e = \{u, v\} \in E, u \in V, u \neq v\} \cup \\ \{&v^\ell_\kappa \mid v \in V, \kappa \in \{0,1\}, \ell \in [v]\}.\end{aligned}$$

For every node $v \in V$ and $\kappa \in \{0, 1\}$, we introduce a set $C^{\text{IP}}_{v_\kappa}$ in $\mathcal{M}$, where

$$C^{\text{IP}}_{v_\kappa} := \{v^e_\kappa, u^e_{\bar{\kappa}} \mid e = \{u, v\} \in E, u \in V, u \neq v\} \cup \{v^\ell_\kappa \mid \ell \in [v]\}.$$

Note that by construction and assumption on I, for every $v \in V$ and $\kappa \in \{0, 1\}$, $|C^{\text{IP}}_{v_\kappa}| = 2\gamma + v$. In the $n \times n$ matrix A in $\Phi(I)$, we define

$$a_{ij} := \begin{cases} 2\gamma + i & \text{if } i = j \\ 2 & \text{otherwise} \end{cases}$$

for all $i, j \in [n]$ with $i \leq j$. In the $n \times n$ matrix B in $\Phi(I)$, we define

$$b_{uv} := \begin{cases} \mathsf{W} & \text{if } u = v \\ \mathsf{w}(e) & \text{otherwise, where } e = \{u, v\} \in E \end{cases}$$

for all $u, v \in [n]$ with $u \leq v$.

Solution Mapping We say that a solution $\mathsf{S} \in F(\Phi(I))$ is *position-consistent* if for each $i \in [n]$, set $C^{\text{IP}}_{v_*}$ on position i has cardinality a_{ii}. If solution $\mathsf{S} \in F(\Phi(I))$ is position-consistent, then function $\Psi(I, \mathsf{S}) := \mathsf{p}$ with $\mathsf{p} : V \to \{0, 1\}$, where $\mathsf{p}(v) := \kappa$ for each $C^{\text{IP}}_{v_\kappa} \in \mathsf{S}$ with $\kappa \in \{0, 1\}$. If S is not position-consistent, then the solution p computed by $\textsc{Init}_{\text{MC}}(I)$ is returned. This terminates the description of the reduction. ○

Lemma 7.16. *Every locally optimal solution $\mathsf{S} \in F(\Phi(I))$ is position-consistent.*

Proof. Assume there exists a locally optimal solution $\mathsf{S}' \in F(\Phi(I))$ which is not position-consistent. This implies that there exists a position $i \in [n]$ such that for set $C^{\text{IP}}_{v_*} \in \mathsf{S}'$ on position i, $|C^{\text{IP}}_{v_*}| \neq a_{ii}$. Exchanging $C^{\text{IP}}_{v_*} \in \mathsf{S}'$ for set $C^{\text{IP}}_{u_*} \in \mathcal{M}$ on position i in S' with $|C^{\text{IP}}_{u_*}| = a_{ii}$ strictly improves the cost of S', since by construction

$$b_{ii} > \sum\nolimits_{j \in [n], j > i} b_{ij} + \sum\nolimits_{\ell \in [n], \ell < i} b_{\ell i}$$

and entries b_{ij} for all $j \in [n], j > i$ and $b_{\ell i}$ for all $\ell \in [n], \ell < i$ are the only terms that may be lost in the cost of S' due to the exchange. A contradiction to S' being locally optimal. □

Lemma 7.17. $\textsc{MaxCut} \leq_{pls}$ IP-(k) *for all $k \geq 1$.*

Proof. Assume there exists a solution $\mathsf{S} \in F(\Phi(I))$ which is locally optimal for $\Phi(I)$, but $\Psi(I, \mathsf{S})$ is not locally optimal for I. By Lemma 7.16, S is position-consistent. Since by assumption, $\Psi(I, \mathsf{S})$ is not locally optimal for I, there exists a node $v \in V$ in instance $I \in D_{\text{MC}}$, which can switch partition such that now edges $e_{i_1}, \ldots, e_{i_p} \in E$ with $i_1, \ldots, i_p, p \in [|E|]$ are in the cut, edges $e_{j_1}, \ldots, e_{j_q} \in E$ with $j_1, \ldots, j_q, q \in [|E|] \setminus \{i_1, \ldots, i_p\}$ are not in the cut and the cost strictly increases by $\Delta > 0$. Now, this implies that in $\Phi(I)$, set $C^{\text{IP}}_{v_\kappa} \in \mathsf{S}$ with $\kappa \in \{0, 1\}$ can be exchanged for set $C^{\text{IP}}_{v_{\bar\kappa}} \in \mathcal{M}$ on position v in S. By construction, for each $e_\ell = \{u, v\}$ with $e_\ell \in \{e_{i_1}, \ldots, e_{i_p}\}$,

$$|C^{\text{IP}}_{u_\kappa} \cap C^{\text{IP}}_{v_{\bar\kappa}}| = 2 = a_{uv}$$

and for each $e_t = \{v, x\}$ with $e_t \in \{e_{j_1}, \ldots, e_{j_q}\}$,

$$|C^{\text{IP}}_{v_{\bar\kappa}} \cap C^{\text{IP}}_{x_{\bar\kappa}}| = 0 \neq a_{vx}.$$

By definition of B, this strictly increases the cost of S by Δ. A contradiction to S being locally optimal. □

7.2.10 On the Complexity of ComparativeContainment-(k)

In this subsection, we prove that ComparativeContainment-(k) is $\mathcal{PLS}$-complete for all $k \geq 1$. Given an instance $I = (\mathcal{C}, \mathcal{X}) \in D_{(h)\text{-CNFSat}}$ for some $h \in \mathbb{N}$, we construct an instance $\Phi(I) = (\mathcal{M}, \mathcal{N}, \mathsf{w}) \in D_{\text{CC-(1)}}$. Here, $\Phi(I)$, consists of two collections $\mathcal{M}$ and $\mathcal{N}$ from a finite set $\mathcal{B}$ and a weight function $\mathsf{w} : \mathcal{M} \cup \mathcal{N} \to \mathbb{N}_0$ that maps sets from collections $\mathcal{M} \cup \mathcal{N}$ to positive integer weights.

The Reduction In a nutshell, the main idea is to *encode every satisfying assignment of a clause via sets* containing the respective literals and possessing the weight of the clause. This is polynomial in the size of the input since the length of each clause in I is at most h. Additionally, we add *large incentives* to *include some literal of each variable* and *medium incentives* to *exclude that both literals of a variable are picked up* in every locally optimal solution.

In more detail, given instance I, we construct an instance $\Phi(I)$ of CC-(1) with the finite set $\mathcal{B} := \{x, \bar{x} \mid x \in \mathcal{X}\}$. For every $x \in \mathcal{X}$, we introduce a set

$$X_x^{\text{CC}} := \{y, \bar{y} \mid y \in \mathcal{X}, y \neq x\}$$

in $\mathcal{N}$ with $\mathsf{w}(X_x^{\text{CC}}) := 2\mathsf{W}$. This terminates the description of $\mathcal{N}$. Now, we define collection $\mathcal{M}$. Let

$$R_x := \{y, \bar{y} \mid y \in \mathcal{X}, y \neq x\}.$$

For every $x \in \mathcal{X}$, we introduce two sets $C_x^{\text{CC}} := R_x \cup \{x\}$ and $C_{\bar{x}}^{\text{CC}} := R_x \cup \{\bar{x}\}$ in $\mathcal{N}$ with $\mathsf{w}(C_x^{\text{CC}}) = \mathsf{w}(C_{\bar{x}}^{\text{CC}}) := \mathsf{W}$. Let

$$I_{C_i} := \{x, \bar{x} \mid x \text{ does not appear in clause } C_i \in \mathcal{C}\}$$

denote the set of all "fan-out" variables, i. e. all variables in $\mathcal{B}$ which are irrelevant for satisfying clause $C_i \in \mathcal{C}$. Extending the technique from Subsection 7.2.7, for every clause $C_i \in \mathcal{C}$, we encode each satisfying assignment for C_i by a set containing the corresponding elements from $\mathcal{B}$, add all variables irrelevant for satisfying the clause, define it to possess the weight of clause C_i and add it to $\mathcal{M}$. In detail, for every clause $C_i(x_{i_1}, \ldots, x_{i_{h_i}}) \in \mathcal{C}$ of length $h_i \leq h$ and weight w_i and each assignment $\mathsf{a} : \{x_{i_1}, \ldots, x_{i_{h_i}}\} \to \{0, 1\}$ which satisfies clause C_i, we introduce a set

$$C_i^{\text{CC}(\mathsf{a})} := I_{C_i} \cup \{\varphi(x_{i_1}), \ldots, \varphi(x_{i_{h_i}})\},$$

in $\mathcal{M}$ with $\mathsf{w}(C_i^{\text{CC}(\mathsf{a})}) := \mathsf{w}_i$, where for all $x \in \{x_{i_1}, \ldots, x_{i_{h_i}}\}$,

$$\varphi(x) := \begin{cases} x & \text{if } \mathsf{a}(x) = 1 \\ \bar{x} & \text{otherwise.} \end{cases}$$

Solution Mapping We slightly modify the definition an element-consistent solution $\mathsf{S} \in F(\Phi(I))$ from Subsection 7.2.8. Here, we call a solution $\mathsf{S} \in F(\Phi(I))$ *element-consistent* if $|\mathsf{S}| = |\mathcal{B}|/2$ and for every $x \in \mathsf{S}$, $\bar{x} \notin \mathsf{S}$. If solution $\mathsf{S} \in F(\Phi(I))$ is element-consistent, then function $\Psi(I, \mathsf{S}) := \mathsf{a}$ with $\mathsf{a} : \mathcal{X} \to \{0, 1\}$, where for all $x \in \mathcal{X}$

$$\mathsf{a}(x) := \begin{cases} 1 & \text{if } x \in \mathsf{S} \\ 0 & \text{otherwise.} \end{cases}$$

If S is not element-consistent, then the assignment a computed by $\textsc{Init}_{\textsc{CNFSat}}(I)$ is returned. This terminates the description of the reduction. ○

Lemma 7.18. *Every locally optimal solution $\mathsf{S} \in F(\Phi(I))$ is element-consistent.*

Proof. Assume there exists a locally optimal solution $\mathsf{S}' \in F(\Phi(I))$ with $|\mathsf{S}'| < |\mathcal{B}|/2$. This implies that there exists an element $x \in \mathcal{B}$ with $x, \bar{x} \notin \mathsf{S}'$. Adding x to S' increases the cost of S', since now $\mathsf{S}' \nsubseteq X_x^{\mathrm{CC}}$. The weight of X_x^{CC} both dominates the sum of the weights lost due to $\mathsf{S}' \nsubseteq C_{\bar{x}}^{\mathrm{CC}}$ and the sum of the weights of the remaining sets in $\mathcal{M}$ in which S' is not entirely contained any more. A contradiction to S' being locally optimal.

Assume there exists a locally optimal solution $\mathsf{S}' \in F(\Phi(I))$ with $|\mathsf{S}'| > |\mathcal{B}|/2$. By pigeonhole principle there exists an element $y \in \mathsf{S}'$ with $\bar{y} \in \mathsf{S}'$. Removing $\bar{y} \in \mathsf{S}'$ increases the cost of S', since, on the one hand, it does not alter any containment of S' in sets from $\mathcal{N}$. On the other hand, $\mathsf{S}' \subseteq C_y^{\mathrm{CC}}$ now and the weight of C_y^{CC} dominates the sum of the smaller weights of sets in $\mathcal{M}$ in which S' is not entirely contained any more. A contradiction to S' being locally optimal.

Assume there exists a locally optimal solution $\mathsf{S}' \in F(\Phi(I))$ such that S' contains two elements $x, \bar{x} \in \mathsf{S}'$. The above cases imply that $|\mathsf{S}'| = |\mathcal{B}|/2$. By pigeonhole principle, there exists an element $y \in \mathcal{B}$ with $y, \bar{y} \notin \mathsf{S}'$. Thus, exchanging x for y increases the cost of S', since now $\mathsf{S}' \nsubseteq X_y^{\mathrm{CC}}$, $\mathsf{S}' \subseteq C_{\bar{x}}^{\mathrm{CC}}$ and still $\mathsf{S}' \nsubseteq X_{\bar{x}}^{\mathrm{CC}}$; the sum of the weights of X_y^{CC} and C_y^{CC} dominates both the sum of the weights lost due to $\mathsf{S}' \nsubseteq C_{\bar{y}}^{\mathrm{CC}}$ and the sum of the smaller weights of sets in $\mathcal{M}$ in which S' is not entirely contained any more. A contradiction to S' being locally optimal. □

Lemma 7.19. (h)-CNFSAT $\leq_{pls}$ CC-(k) *for all* $k \geq 1$.

Proof. Assume there exists a solution $\mathsf{S} \in F(\Phi(I))$ which is locally optimal for $\Phi(I)$, but $\Psi(I, \mathsf{S})$ is not locally optimal for I. By Lemma 7.18, S is element-consistent. Since $\mathsf{a} := \Psi(I, \mathsf{S})$ is not locally optimal for I, there exists a variable $x \in \mathcal{X}$ in instance $I \in D_{(h)\text{-CNFSAT}}$, which can be flipped such that in the resulting solution $\mathsf{a}' \in F(I)$, clauses $C_{i_1}, \ldots, C_{i_p} \in \mathcal{C}$ with $i_1, \ldots, i_p, p \in [|\mathcal{C}|]$ become satisfied, clauses $C_{j_1}, \ldots, C_{j_q} \in \mathcal{C}$ with $j_1, \ldots, j_q, q \in [|\mathcal{C}|] \setminus \{i_1, \ldots, i_p\}$ become unsatisfied and the cost strictly increases by $\Delta > 0$. Without loss of generality, let $x \in \mathsf{S}$. Now, this implies that in $\Phi(I)$, element $x \in \mathsf{S}$ can be exchanged for element $\bar{x} \in \mathcal{B}$; call the resulting solution S'. Note that $\mathsf{S} \subseteq X_*^{\mathrm{CC}}$ if and only if $\mathsf{S}' \subseteq X_*^{\mathrm{CC}}$ for all $X_*^{\mathrm{CC}} \in \mathcal{N}$. As in

Subsection 7.2.7, for some clause $C_i \in \mathcal{C}$, denote by a'_i the assignment of a' restricted to the set of variables in C_i. Now, S' is entirely contained in each set

$$C_{i_1}^{\mathrm{CC}(\mathsf{a}'_{i_1})}, \ldots, C_{i_p}^{\mathrm{CC}(\mathsf{a}'_{i_p})} \in \mathcal{M},$$

and S' is not entirely contained in each set

$$C_{j_1}^{\mathrm{CC}(\mathsf{a}'_{j_1})}, \ldots, C_{j_q}^{\mathrm{CC}(\mathsf{a}'_{j_q})} \in \mathcal{M};$$

furthermore, $\mathsf{S}' \nsubseteq C_x^{\mathrm{CC}}$ and $\mathsf{S}' \subseteq C_{\bar{x}}^{\mathrm{CC}}$. By definition of w, this strictly increases the cost of S by Δ. A contradiction to S being locally optimal. $\square$

7.2.11 Proving the Tightness of the Reductions

In this subsection, we consider the tightness of all reductions presented in Subsections 7.2.2–7.2.10. We prove the following lemma:

Lemma 7.20. *All reductions presented in Subsections 7.2.2–7.2.10 are tight.*

Proof. First, consider the reduction for SetSplitting-(k) in Subsection 7.2.4. When defining $\mathscr{R}$ as $F_{\mathrm{SSp}\text{-}(1)}(\Phi(I))$ for a given instance $I \in D_{\mathrm{MC}}$, it is obvious to see that our reduction is tight.

Now, let P be a weighted standard set problem for which we show intractability in Subsections 7.2.2, 7.2.3 and 7.2.5–7.2.10 and let I be an instance of P. We define the set $\mathscr{R}_P$ to be the set of all standard or consistent solutions in $F_P(\Phi(I))$, where the definition of a standard or consistent solution is with respect to P. As shown in the respective subsections, $\mathscr{R}_P$ contains the set of all locally optimal solutions in $F_P(\Phi(I))$. An improving step in the given neighborhood leads from a standard or consistent solution $\mathsf{S} \in F_P(\Phi(I))$ to a standard or consistent solution $\mathsf{S}' \in F_P(\Phi(I))$. Solution $\Psi(I, \mathsf{S}')$ is a better neighbor of $\Psi(I, \mathsf{S})$ and can only differ in the assignment of a single variable, since, otherwise, this would require a larger neighborhood than the one given. $\square$

7.2.12 On the Tractability of Weighted Standard Set Problems

Subsections 7.2.2–7.2.10 yield the following intractability results for weighted standard set problems, as introduced in Section 2.2.4:

Theorem 7.1. *The problems* SSp-(k), TS-(k), HS-(k), SB-(k), IP-(k), *and* CC-(k) *are tight $\mathcal{PLS}$-complete for all $k \geq 1$. The problems* SP-(k) *and* SC-(k) *are tight $\mathcal{PLS}$-complete for all $k \geq 2$. The problems* W3DM-(k, ℓ) *and* X3C-(k) *are tight $\mathcal{PLS}$-complete for all $k \geq 6$ and $\ell \geq 12$.*

Subsections 7.2.3 and 7.2.5 provide the following two tractability results for SetPacking and SetCover:

Theorem 7.2. *The problems* SP-(1) *and* SC-(1) *are polynomial-time solvable.*

7.3 Conclusion and a Discussion of Question 7

Set problems are fundamental combinatorial optimization problems with a wide range of applications. In this chapter, we studied the tractability of computing locally optimal solutions for weighted standard set problems. For almost all problems we investigated, the neighborhood structure is superimposed by the *k-differ neighborhood*. Here, two solutions are mutual neighbors if they differ in at most k elements which describe a solution. Before stepping into the $\mathcal{PLS}$ reductions, we presented the general technique of our intractability proofs in Section 7.1.

Results Obtained For most of the weighted standard set problems we investigated, we showed the *intractability* of computing locally optimal solutions for the respective *1-differ neighborhood* in Section 7.2. This means that the problems are already hard, when one element describing the solution is allowed to be added, deleted, or exchanged for another element which is not part of the solution. For the problems SetPacking and SetCover, we derived *tight bounds* on the tractability of computing locally optimal solutions for the respective *2-differ neighborhood* in Subsection 7.2.3 and Subsection 7.2.5. This means that SetPacking-(1) (resp. SetCover-(1)) is polynomial-time computable, whereas SetPacking-(k) (resp. SetCover-(k)) is tight $\mathcal{PLS}$-complete for all $k \geq 2$. Furthermore, we proved in Subsection 7.2.2 that the problems Weighted-3-DimensionalMatching and Exact-Cover-By-3-Sets are already $\mathcal{PLS}$-complete for a smaller neighborhood than previously known [30].

Discussion of Question 7 Considering Question 7, we were able to present the non-sophisticated polynomial-time algorithms GreedyPacking and GreedyCover. These algorithms were sufficient to prove the tractability of computing locally optimal solutions for SetPacking-(1) and SetCover-(1). Let us stress that GreedyPacking and GreedyCover were designed without bearing the quality of the obtained solutions in mind, but only focusing on the tractability of the respective problems. For all remaining weighted standard set problems we investigated, we were able to show their intractability, as outlined in the previous paragraph. According to the concept of $\mathcal{PLS}$-completeness, the intractability of the corresponding weighted standard set problems suggests that there is little hope for polynomial-time algorithms for these hard problems. Moreover, our investigations suggest that—up to a few exceptions—$\mathcal{PLS}$-completeness seems to be the natural behavior of computing locally optimal solutions for weighted standard set problems. This indicates that computing locally optimal solutions for these problems via successive improvements may not yield a sufficient performance improvement over computing globally optimal solutions, in general. Our analysis also unveils that the hardness of the problems we investigated stems from a combination of a numerical problem on an underlying combinatorial problem.

Open Problems Despite this insight, the exact tractability of Weighted-3-DimensionalMatching-(k, ℓ) and Exact-Cover-By-3-Sets-(k) for $k < 6$ or $\ell < 12$

remains unsettled. In general, our investigation can only serve as a starting point to sharpen the boundary on the tractability of weighted standard set problems, as intensively studied in the literature. To the best of our knowledge, the results we present are one of the very few $\mathcal{PLS}$ results for local search on weighted standard set problems. Our knowledge of their tractability is rather limited and not at all comparable with the rich knowledge we have about computing globally optimal solutions for standard set problems, as given in the book of Garey and Johnson [40].

Chapter 8

Conclusion and Directions for Further Research

In this chapter, we present a general conclusion which recapitulates the problems investigated and the results obtained in the course of this thesis. Additionally, we outline directions for further research. In Section 8.1, we first summarize the results we presented in the course of this thesis and discuss Questions 1–3. In Section 8.2, we present open problems related not only to the problems we studied in this thesis, but also from the field of local search in general. In Section 8.3, we close this thesis with some remarks on our knowledge of the tractability of computing of locally optimal solutions we would like the reader to take from this thesis.

8.1 General Conclusion and a Discussion of Questions 1–3

In this section, we discuss Questions 1–3, given in Subsection 1.3.1. We first pass in review the problems for which we were able to settle the complexity of computing a locally optimal solution, as asked for in Question 1. We proceed with a bird's-eye view of our reductions, identifying commonalities between them, as inquired in Question 2. We close with the potential sources of intractability of the $\mathcal{PLS}$ problems which we identified as hard, as asked for in Question 3.

8.1.1 Settling the $\mathcal{PLS}$-Complexity of the Problems Considered in this Thesis

In this thesis, we studied the tractability of computing locally optimal solutions for problems arising in the fields of *game theory* and *optimization.* For our investigation, we used the framework of $\mathcal{PLS}$, as introduced by Johnson, Papadimtriou, and Yannakakis [56].

1. In *game theory*, congestion games are a widely accepted model to investigate the behavior and performance of large-scale distributed networks with autonomous participants. The class of *restricted network congestion games* which we studied in Chapter 5 is a subclass of congestion games where for each player there exists a set of edges which he is not allowed to use. Rosenthal's potential

function guarantees the existence of a Nash equilibrium, as local minima of the potential function coincide with Nash equilibria; moreover, Rosenthal's potential function is polynomial-time computable. This allows to formulate the problem of computing a Nash equilibrium in a given restricted network congestion game as a $\mathcal{PLS}$ problem. The input consists of a restricted network congestion game and the neighborhood structure is superimposed by the SelfishSteps algorithm. Our results yielded a surrounding intractability answer to this $\mathcal{PLS}$ problem. Not only is the problem of *computing a Nash equilibrium* in a *restricted network congestion game* involving *two players* only *$\mathcal{PLS}$-complete*, it also has the *all-exp property.* This means that there exist instances and initial states of the games, such that every sequence of selfish steps has exponential length. The results hold for directed and undirected networks. Notably, our results are optimal, since for one player, computing a Nash equilibrium reduces to finding a shortest path. Hence, our results settle the complexity of computing Nash equilibria in restricted network congestion games.

2. From the field of *optimization*, we investigated the complexity of computing locally optimal solutions for the Maximum Constraint Assignment problem in Chapter 6 and for *weighted standard set problems* in Chapter 7.

   The Maximum Constraint Assignment (in short MCA) problem is a local search version of weighted Generalized Maximum Satisfiability on constraints over variables with higher valence. Two solutions are mutual neighbors, if they differ in the assignment of a single variable. In Chapter 6, we focused on the subclass (p, q, r)-MCA$_{k\text{-par}}$, where each constraint has length at most p, each variable appears in at most q constraints and takes at most r values; additionally, the set of constraints is k-partite. We extended and refined a technique from Krentel [67] and showed that $(3, 2, 3)$-MCA$_{3\text{-par}}$ and $(2, 3, 6)$-MCA$_{2\text{-par}}$ are *tight $\mathcal{PLS}$-complete.* When neglecting the valence of the variables, our results are optimal, since $(2, 2, r)$-MCA is polynomial-time computable for every $r \in \mathbb{N}$. Additionally, we obtained that for the special case of binary variables, $(6, 2, 2)$-MCA is *tight $\mathcal{PLS}$-complete.*

   The *weighted standard set problems* we considered in Chapter 7 are local search versions of their respective decision problems, as intensively studied in the literature. For most of these problems, we used the k-differ neighborhood, where two solutions are mutual neighbors if they differ in at most k elements which describe a solution. Our investigations unveiled that *up to a few exceptions, all weighted standard set problems we considered* are *tight $\mathcal{PLS}$-complete*, even for the 1-differ neighborhood. For Weighted-3-DimensionalMatching and Exact-Cover-By-3-Sets, we obtained intractability results for a smaller neighborhood than previously known, where we are not certain if our results are optimal. For the problems SetPacking and SetCover with the 1-differ neighborhood, we presented polynomial-time algorithms; again for the 2-differ neighborhood, we were able to provide tight $\mathcal{PLS}$ reductions from $(3, 2, 3)$-MCA and (3, 2, 3)-MinCA, respectively.

The hardness results presented in the course of this thesis *demarcate the tractability* of computing locally optimal solutions for the respective problems, up to a few exceptions. The intractability results we obtained suggest that there is little hope for polynomial-time algorithms for the corresponding problems. Moreover, computing locally optimal solutions for intractable problems via successive improvements may not yield a sufficient performance improvement over computing globally optimal solutions, in general. From our results, we get the feeling that $\mathcal{PLS}$-completeness is the natural behavior of many local search problems arising from intractable optimization problems.

We would like to stress that we put an *emphasis on proving the tightness* of all the reductions we presented in the course of this thesis. We believe that the tight $\mathcal{PLS}$-completeness of all problems we proved to be intractable—and especially the rather technically involved tight $\mathcal{PLS}$-completeness of the Maximum Constraint Assignment problems in Chapter 6—might prove useful in establishing that other problems are tight $\mathcal{PLS}$-complete.

8.1.2 A Note on the General Structure of Our $\mathcal{PLS}$ Reductions

At first glance, the reductions we presented in the course of this thesis tend to be rather *problem-specific constructions*. They mostly involved *gadgets* that exploit certain characteristics of the $\mathcal{PLS}$ problem under investigation. For example, consider the assign$(*, *)$ gadget, presented in Subsection 7.2.2. This gadget was a crucial part of our construction to prove the intractability of Weighted-3-DimensionalMatching (resp. Exact-Cover-By-3-Sets). With the help of the assign$(*, *)$ gadget, we created a *framework* which eventually ensured that the variable assignments induced by each locally optimal matching (resp. covering) are consistent. In the design of the gadget, we exploited a characteristic of the problem that in each feasible solution, every element can only be contained in a single triple (resp. set). At first, it might seem hard to imagine how to reuse that problem-specific knowledge in other reductions which involve problems from potentially different areas.

Nonetheless, all reductions from $\mathcal{PLS}$ problem A to some $\mathcal{PLS}$ problem B which we presented share some commonalities. Let $I \in D_A$ be an instance of A and denote $\Phi(I) \in D_B$ an instance of B, which we constructed in the course of this thesis. We attempt to identify and categorize the *common fragments* of all our reductions by *weight*. In spirit with the seminal work of Krentel [67], we identify these weights to be either *large*, *medium*, or *small*:

1. Note that all of our rather technically involved reductions used some sort of construction to create a *framework* to simulate I in $\Phi(I)$. Here, *gadgets with large weights* ensured that locally optima solutions in $\Phi(I)$ correspond to locally optimal solutions in I. Typically, a single large weight exceeded the sum of all weights on all lower levels. A framework can, for example, ensure the consistency

of the variable assignment or the partition of the set of nodes in I which is induced by a solution in $F_B(\Phi(I))$, as in most reductions in Chapter 7. Note that the reduction for SetSplitting-(1) presented in Chapter 7 lacks such a construction, since the reduction presented there is straightforward. In our reductions for (2)-RDNCG and (2)-RUNCG in Chapter 5, one construction created a framework involving large weights which ensured that the choices of strategies of the two players induced assignments, which did not differ in more than one bit. In our reductions for $(3,2,3)$-MCA$_{\text{3-par}}$ and $(2,3,6)$-MCA$_{\text{2-par}}$ in Chapter 6, the predicates for gates in circuit S^0 and S^1 created a framework which ensured that the computation of circuits S^0 and S^1 was always correct, with respect to the ternary logic used in the construction. Let us stress that all of the above described constructions are carefully designed such that for every solution which is not locally optimal, there exists a sequence of improving steps that terminates within the framework. The size of the weights allows to neglect all weights on the lower levels in an improving step on the largest level, by construction.

2. In most of our reductions, we simulated instance I in $\Phi(I)$, using a *problem-specific construction* which involved *medium weights*. If present, the framework created in item 1 ensured that locally optimal solutions in $\Phi(I)$ have certain structural properties. By construction, we stored the *value of the objective function* of a solution of the given instance I on this level. Note that the value of the objective function throttled the search for a better solution, in case the objective function could be improved. This was the *core* of all our reductions.

3. Additional constructions involving *small weights* gave *modest incentives* in case the framework on the largest level outlined in item 1 was not fully rolled out or the entire simulation of instance I in $\Phi(I)$ needed to be nudged in a certain direction. For example, in our reduction for SetPacking-(2) in Chapter 7, auxiliary dummy sets of small weight served as "escape options" in case the sets of a solution of $\Phi(I)$ intersected. They allowed to replace an intersecting set with an intersection-free dummy set at a minimal reward. Afterwards, dummy sets could again be safely removed from the solution, such that locally optimal solutions in $\Phi(I)$ induced locally optimal solutions in I. In our reductions for $(3,2,3)$-MCA$_{\text{3-par}}$ and $(2,3,6)$-MCA$_{\text{2-par}}$ in Chapter 6, all predicates of small weight played a vital part in nudging circuits S^0, S^1, and the comparator in the direction of the improved solution. Once the large weights caught up, the construction on the small level was reset by the medium weights for the next run.

The *structure of our proofs* mostly followed the layout of the weights in a natural way. We first showed that each locally optimal solution obeys the framework installed in item 1. We usually identified such solutions as standard solutions or to be consistent for some property. In the actual proof of $\mathcal{PLS}$-completeness, we could then solely focus on standard or consistent solutions.

Let us remark that most of the $\mathcal{PLS}$-completeness proofs presented in the literature that we are aware of follow this approach with minor modifications. Nonetheless, we should mention that Skopalik and Vöcking [98] present a *slightly different general layout*. In their paper [98], the authors present a reduction which implements a *binary counter* on the *largest level*. All succeeding levels then handle the simulation of the given input problem. By construction, the authors can guarantee that the potential stored on the counter is at least the largest possible improvement of the potential function for the problem to be simulated.

8.1.3 Possible Sources of Intractability in Our $\mathcal{PLS}$-Complete Problems

In general, the tractability of a $\mathcal{PLS}$ problem crucially depends on both the *range of the involved numbers* and the *structure of the neighborhood*.

1. If all involved numbers are polynomially bounded, then locally optimal solutions can be computed via successive improvements in polynomial time, regardless of the neighborhood structure. Note that for all intractability results we provided in this thesis, the range of the involved numbers is *exponential*.
2. Most of the neighborhood structures of the problems we studied in this thesis are rather non-involved. Yet, we were able to show intractability for these simple neighborhoods. Only in the special cases of SetPacking and SetCover with the 1-differ neighborhood, could we capitalize on the neighborhood structure with greedy algorithms. In both cases, the neighborhood structure limited the options for improvements in every step such that the problems became easy, regardless of the range of the weights.

The $\mathcal{PLS}$-completeness proofs we presented in the course of this thesis indicate that the *intractability* of a $\mathcal{PLS}$ problem we investigated *stems from a combination of a numerical problem on an underlying combinatorial problem*. In restricted network congestion games, which we investigated in Chapter 5, the numerical problem is embedded in the delay functions of the resources and the combinatorial problem is given by the structure of the underlying network. Let us remark that our construction in Chapter 5 heavily exploited the potential complexity of the underlying network. We designed strategies which were intensively interwoven, yet each player still simulated an assignment of nodes to partitions, node-by-node, in his exclusive subnetwork. Transforming this construction to standard network congestion games was one of the obstacles we could not overcome in order to extend our result. In the Maximum Constraint Assignment problem, which we studied in Chapter 6, the numerical problem is embedded in the weights of the predicates which define constraints. We get the impression that the combinatorial problem arises from the rich set of defining satisfying assignments for predicates along with the interconnection of predicates and constraints themselves. In local search versions of weighted standard set problems, our analysis in Chapter 7 unveils that the numerical problem is embedded in the weight function on the set system and the combinatorial problem is encoded in the structure of the sets.

8.2 Open Problems

In this section, we present an excerpt of additional open problems, besides the ones we presented in the course of this thesis. We categorize them as problems from *combinatorial optimization*, *game theory*, and *smoothed complexity*.

8.2.1 Combinatorial Optimization

As outlined in Section 3.1, several hardness results have been established for $\mathcal{PLS}$ problems arising from combinatorial optimization. While these results show that the problems are in general $\mathcal{PLS}$-complete, the *exact* bounds on the $\mathcal{PLS}$-complexity of the problems are still unknown. For MaxCut, we outlined in Section 3.1 that the problem is $\mathcal{PLS}$-complete for unbounded degree and polynomial-time solvable for maximum degree three. While the result of Krentel [67], implies that MaxCut is $\mathcal{PLS}$-complete for *some* fixed maximum degree, the minimum degree required for the problem to be $\mathcal{PLS}$-complete is still unknown. A recent result by Tscheuschner [104] shows that a maximum degree of five is sufficient such that MaxCut is $\mathcal{PLS}$-complete. Yet, the exact complexity remains unsettled as to the best of our knowledge we are not aware of any tractability or intractability results for MaxCut with maximum degree four. Similarly, for the Traveling Salesman Problem where the neighborhood structure is superimposed by the well-known k-Opt algorithm, the only published results imply the $\mathcal{PLS}$-completeness for $k \gg 1,000$. The exact complexity for $2 \leq k \ll 1,000$ is still unsettled. Considering the Simplex algorithm, the question arises, if there exists a pivot rule which guarantees a polynomially upper-bounded number of steps of the Simplex algorithm.

8.2.2 Game Theory

Section 3.1 outlines that except for special cases, general congestion games are $\mathcal{PLS}$-complete. For a thorough understanding of these games, determining the source of the inherent complexity would be beneficial. To this extent, the precise impact of the players, the resources, and their mutual interaction on the hardness are yet to be determined. For example, the exact complexity of network congestion games with a finite number of players is still unknown. As outlined in Section 3.2 and settled in Chapter 5, exact results only exist for the special case of restricted network congestion games. In the simplest model of singleton congestion games and arbitrary non-decreasing latency functions, no results are known, considering the complexity of computing a Nash equilibrium. Turning to approximation, the complexity of computing δ-approximate Nash equilibria for symmetric congestion games, which do not satisfy some smoothness condition is still unsettled. Turning to coalitions in the symmetric singleton congestion game model, there remains a gap on the complexity of computing a Nash equilibrium. For coalitions of size one, the problem is known to be computable in polynomial time, while if users may form arbitrary non-fixed coalitions of size at least eight, the problem becomes $\mathcal{PLS}$-complete [27].

8.2.3 Smoothed Complexity

The concept of $\mathcal{PLS}$-completeness implies that computing a locally optimal solution of *some* $\mathcal{PLS}$-complete problem is as hard as computing a locally optimal solution of *any* problem in the class $\mathcal{PLS}$. Loosely speaking, while this states that there exist instances on which local search algorithms take exponential time, what is the smoothed complexity of $\mathcal{PLS}$-complete problems? Let us remark that smoothed complexity results only exist for problems from Section 1.2.2; for all other problems outlined in this thesis, especially for MaxCut and congestion games, we are not aware of any results considering their smoothed complexity.

8.3 What is There to Take Home?

Local search is a standard approach to approximate solutions of hard combinatorial optimization problems which has proven to be successful in a wide range of areas over the last decades. The framework of $\mathcal{PLS}$ was introduced to theoretically investigate the complexity of local search problems and drew additional attention from game theory in recent years. In general, our knowledge about the class $\mathcal{PLS}$ is currently rather limited and by far not comparable with the rich knowledge which we have about the class $\mathcal{NP}$. As we have outlined and contributed to in the course of this thesis, the complexity of a handful of $\mathcal{PLS}$ problems has been settled; for numerous other problems, determining their complexity remains tantalizingly open.

Lists

List of Figures

List of Tables

Bibliography

Remark: Each entry is followed by a list of pages which refer to the publication.

[1] E. Aarts and J. K. Lenstra, editors. *Local Search in Combinatorial Optimization.* John Wiley & Sons, Inc., New York, NY, USA, 1997. ISBN 0471948225. 7, 31

[2] E. Aarts, J. Korst, and W. Michiels. *Theoretical Aspects of Local Search (Monographs in Theoretical Computer Science. An EATCS Series).* Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2007. ISBN 3540358536. 19, 31

[3] S. Abramsky, C. Gavoille, C. Kirchner, F. M. auf der Heide, and P. G. Spirakis, editors. *Proceedings of the 37th International Colloquium on Automata, Languages and Programming (ICALP 2010), Part I*, volume 6198 of *Lecture Notes in Computer Science*, 2010. Springer. ISBN 978-3-642-14164-5. 149, 151

[4] H. Ackermann and A. Skopalik. On the Complexity of Pure Nash Equilibria in Player-Specific Network Congestion Games. In X. Deng and F. C. Graham, editors, *Proceedings of the 3rd International Workshop on Internet and Network Economics (WINE 2007)*, volume 4858 of *Lecture Notes in Computer Science*, pages 419–430. Springer, 2007. ISBN 978-3-540-77104-3. 15, 25, 36, 45, 47, 52, 108

[5] H. Ackermann, H. Röglin, and B. Vöcking. On the Impact of Combinatorial Structure on Congestion Games. *Journal of the ACM (JACM)*, 55(6):1–22, 2008. ISSN 0004-5411. 34, 35

[6] I. Adler, R. M. Karp, and R. Shamir. A Simplex Variant Solving an $m \times d$ Linear Program in $\mathcal{O}(\min\{m^2, d^2\})$ Expected Number of Pivot Steps. *Journal of Complexity*, 3(4):372–387, 1987. 10

[7] E. Alekseeva, Y. Kochetov, and A. Plyasunov. Complexity of Local Search for the p-Median Problem. *European Journal of Operational Research*, 191(3):736 – 752, 2008. ISSN 0377-2217. 31

[8] E. Angel. A Survey of Approximation Results for Local Search Algorithms. In E. Bampis, K. Jansen, and C. Kenyon, editors, *Efficient Approximation and Online Algorithms*, volume 3484 of *Lecture Notes in Computer Science*, pages 30–73. Springer Berlin / Heidelberg, 2006. 7, 31

[9] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy. Proof Verification and Hardness of Approximation Problems. In *Proceedings of the 33rd Annual Symposium on Foundations of Computer Science (FOCS 1992)*, pages 14–23. IEEE Computer Society, 1992. 6

[10] D. Arthur, B. Manthey, and H. Röglin. k-Means has Polynomial Smoothed Complexity. *The Computing Research Repository (CoRR)*, abs/0904.1113, 2009. 10

[11] G. Ausiello, P. Crescenzi, V. Kann, Marchetti-Spaccamela, G. Gambosi, and A. M. Spaccamela. *Complexity and Approximation: Combinatorial Optimization Problems and Their Approximability Properties.* Springer, January 2000. ISBN 3540654313. 5

[12] L. Babai, editor. *Proceedings of the 36th Annual ACM Symposium on Theory of Computing (STOC 2004)*, 2004. ACM. ISBN 1-58113-852-0. 148

[13] E. Balas and M. W. Padberg. Set Partitioning: A Survey. *SIAM Review*, 18(4): 710–760, 1976. 4

[14] A. Biere, M. Heule, H. van Maaren, and T. Walsh, editors. *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*, 2009. IOS Press. ISBN 978-1-58603-929-5. 5, 6

[15] R. E. Bixby. Solving Real-World Linear Programs: A Decade and More of Progress. *Operations Research*, 50(1):3–15, 2002. 7, 9

[16] K. H. Borgwardt. Probabilistic Analysis of Simplex Algorithms. *Contemporary Mathematics*, 114:21–34, 1990. 10

[17] F. Brandt, F. Fischer, and M. Holzer. Symmetries and the Complexity of Pure Nash Equilibrium. *Journal of Computer and System Sciences (JCSS)*, 75(3): 163 – 177, 2009. ISSN 0022-0000. 31

[18] B. Chandra and M. M. Halldórsson. Greedy Local Improvement and Weighted Set Packing Approximation. In *Proceedings of the Tenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 1999)*, pages 169–176. ACM/SIAM, 1999. 121, 122

[19] B. Chandra, H. J. Karloff, and C. A. Tovey. New Results on the Old k-Opt Algorithm for the Traveling Salesman Problem. *SIAM Journal on Computing (SICOMP)*, 28(6):1998–2029, 1999. 9, 10

[20] P. Chapdelaine and N. Creignou. The Complexity of Boolean Constraint Satisfaction Local Search Problems. *Annals of Mathematics and Artificial Intelligence*, 43(1-4):51–63, 2005. ISSN 1012-2443. 33

[21] X. Chen, X. Deng, and S.-H. Teng. Settling the Complexity of Computing Two-Player Nash Equilibria. *Journal of the ACM (JACM)*, 56(3), 2009. 38

[22] S. Chien and A. Sinclair. Convergence to Approximate Nash Equilibria in Congestion Games. *Games and Economic Behavior*, In Press, Accepted Manuscript, 2009. 35

[23] S. A. Cook. The Complexity of Theorem-Proving Procedures. In *Proceedings of the 3rd Annual ACM Symposium on Theory of Computing (STOC 1971)*, pages 151–158. ACM, 1971. 5

[24] G. Danzig. Programming in Linear Structure. Technical report, U.S. Air Force Comptroller, USAF, Washington, D.C., 1948. 9

[25] C. Daskalakis, P. W. Goldberg, and C. H. Papadimitriou. The Complexity of Computing a Nash Equilibrium. *SIAM Journal on Computing (SICOMP)*, 39 (1):195–259, 2009. 38

[26] R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern Classification.* John Wiley & Sons, Inc., 2000. 7, 10

[27] D. Dumrauf and B. Monien. On the Road to $\mathcal{PLS}$-Completeness: 8 Agents in a Singleton Congestion Game. In C. H. Papadimitriou and S. Zhang, editors, *Proceedings of the 4th International Workshop on Internet and Network Economics (WINE 2008)*, volume 5385 of *Lecture Notes in Computer Science*, pages 94–108. Springer, 2008. ISBN 978-3-540-92184-4. 37, 108, 141

[28] D. Dumrauf and B. Monien. Computing Nash Equilibria for Two-Player Restricted Network Congestion Games is $\mathcal{PLS}$-Complete. *submitted to Parallel Processing Letters*, 2010. 17

[29] D. Dumrauf and T. Süß. On the Complexity of Local Search for Weighted Standard Set Problems. In F. Ferreira, B. Löwe, E. Mayordomo, and L. M. Gomes, editors, *Proceedings of the 6th Conference on Computability in Europe (CiE 2010): Programs, Proofs, Processes*, volume 6158 of *Lecture Notes in Computer Science*, pages 132–140. Springer, 2010. ISBN 978-3-642-13961-1. 17, 27, 28, 29

[30] D. Dumrauf, B. Monien, and K. Tiemann. MultiProcessor Scheduling is $\mathcal{PLS}$-Complete. In *Proceedings of the 42nd Hawaii International International Conference on Systems Science (HICSS-42 2009)*, pages 1–10. IEEE Computer Society, 2009. 26, 110, 112, 113, 133

[31] D. Dumrauf and B. Monien. On the $\mathcal{PLS}$-Complexity of Maximum Constraint Assignment. *submitted to Theoretical Computer Science*, 2010. 17, 25, 26, 108

[32] J. Dunkel and A. S. Schulz. On the Complexity of Pure-Strategy Nash Equilibria in Congestion and Local-Effect Games. *Mathematics of Operations Research*, 33(4):851–868, 2008. 34

[33] M. Englert, H. Röglin, and B. Vöcking. Worst Case and Probabilistic Analysis of the 2-Opt Algorithm for the TSP. *Electronic Colloquium on Computational Complexity (ECCC)*, 13(092), 2006. 9, 10

[34] A. Fabrikant, C. H. Papadimitriou, and K. Talwar. The Complexity of Pure Nash Equilibria. In Babai [12], pages 604–612. ISBN 1-58113-852-0. 15, 34

[35] R. Feldmann, M. Gairing, T. Lücking, B. Monien, and M. Rode. Nashification and the Coordination Ratio for a Selfish Routing Game. In J. C. M. Baeten, J. K. Lenstra, J. Parrow, and G. J. Woeginger, editors, *Proceedings of the 30th International Colloquium on Automata, Languages and Programming (ICALP 2003)*, volume 2719 of *Lecture Notes in Computer Science*, pages 514–526. Springer Verlag, 2003. 37

[36] D. Fotakis, S. C. Kontogiannis, E. Koutsoupias, M. Mavronicolas, and P. G. Spirakis. The Structure and Complexity of Nash Equilibria for a Selfish Routing Game. In P. Widmayer, F. T. Ruiz, R. M. Bueno, M. Hennessy, S. Eidenbenz, and R. Conejo, editors, *Proceedings of the 29th International Colloquium on Automata, Languages and Programming (ICALP 2002)*, volume 2380 of *Lecture Notes in Computer Science*, pages 123–134. Springer, 2002. ISBN 3-540-43864-5. 37

[37] D. Fotakis, S. Kontogiannis, and P. Spirakis. Selfish Unsplittable Flows. *Theoretical Computer Science*, 348(2-3):226–239, 2005. ISSN 0304-3975. 13

[38] M. Gairing, T. Lücking, M. Mavronicolas, and B. Monien. Computing Nash Equilibria for Scheduling on Restricted Parallel Links. In Babai [12], pages 613–622. ISBN 1-58113-852-0. 37

[39] M. Gairing, B. Monien, and K. Tiemann. Routing (Un-) Splittable Flow in Games with Player-Specific Linear Latency Functions. In M. Bugliesi, B. Preneel, V. Sassone, and I. Wegener, editors, *Proceedings of the 33rd International Colloquium on Automata, Languages and Programming, (ICALP 2006), Part I*, volume 4051 of *Lecture Notes in Computer Science*, pages 501–512. Springer, 2006. ISBN 3-540-35904-4. 36

[40] M. R. Garey and D. S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness.* Mathematical Sciences Series. W. H. Freeman & Co., New York, NY, USA, 1990. ISBN 0-7167-1045-5. vi, 2, 3, 4, 6, 16, 22, 23, 25, 26, 33, 134

[41] M. X. Goemans, V. S. Mirrokni, and A. Vetta. Sink Equilibria and Convergence. In *Proceedings of the 46th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2005)*, pages 142–154. IEEE Computer Society, 2005. ISBN 0-7695-2468-0. 34

[42] O. Goldschmidt, D. S. Hochbaum, and G. Yu. A Modified Greedy Heuristic for the Set Covering Problem with Improved Worst Case Bound. *Information Processing Letters*, 48(6):305–310, 1993. 121

[43] T. F. Gonzalez. *Handbook of Approximation Algorithms and Metaheuristics (Chapman & Hall/CRC Computer & Information Science Series)*. Chapman & Hall/CRC, 2007. ISBN 1584885505. 6

[44] R. L. Graham. Bounds on Multiprocessing Timing Anomalies. *SIAM Journal on Applied Mathematics*, 17(2):416–429, mar 1969. ISSN 0036-1399. 37

[45] J. Gu. Local Search for Satisfiability (SAT) Problem. *IEEE Transactions on Systems, Man, and Cybernetics*, 23(4):1108–1129, 1993. 6

[46] J. Gu, P. W. Purdom, J. Franco, and B. W. Wah. Algorithms for the Satisfiability (SAT) Problem: A Survey. In D.-Z. Du, J. Gu, and P. Pardalos, editors, *Satisfiability Problem: Theory and Applications*, DIMACS Series in Discrete Mathematics and Theoretical Computer Science, pages 19–152. American Mathematical Society, 1997. 5, 6

[47] M. M. Halldórsson. Approximating Discrete Collections via Local Improvements. In *Proceedings of the Sixth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 1995)*, pages 160–169. ACM/SIAM, 1995. 122

[48] T. Harks and M. Klimm. On the Existence of Pure Nash Equilibria in Weighted Congestion Games. In Abramsky et al. [3], pages 79–89. ISBN 978-3-642-14164-5. 13

[49] J. Håstad. Some Optimal Inapproximability Results. *Journal of the ACM (JACM)*, 48(4):798–859, 2001. 6, 122

[50] E. Hazan, S. Safra, and O. Schwartz. On the Complexity of Approximating k-Set Packing. *Computational Complexity*, 15(1):20–39, 2006. 122

[51] D. S. Hochbaum, editor. *Approximation Algorithms for NP-hard Problems*. PWS Publishing Co., Boston, MA, USA, 1997. ISBN 0-534-94968-1. 6

[52] K. L. Hoffman and M. Padberg. Set Covering, Packing and Partitioning Problems. In C. A. Floudas and P. M. Pardalos, editors, *Encyclopedia of Optimization*, pages 3482–3486. Springer, 2009. ISBN 978-0-387-74758-3. 4, 6

[53] C. A. J. Hurkens and A. Schrijver. On the Size of Systems of Sets Every t of Which Have an SDR, with an Application to the Worst-Case Ratio of Heuristics for Packing Problems. *SIAM Journal on Discrete Mathematics*, 2(1):68–72, 1989. 122

[54] A. K. Jain, M. N. Murty, and P. J. Flynn. Data Clustering: A Review. *ACM Computing Surveys*, 31(3):264–323, 1999. 7

[55] D. S. Johnson and L. A. McGeoch. The Traveling Salesman Problem: A Case Study. In E. H. L. Aarts and J. K. Lenstra, editors, *Local Search in Combinatorial Optimization*, pages 215–310. Wiley and Sons, New York, 1997. 9, 32

[56] D. S. Johnson, C. H. Papadimtriou, and M. Yannakakis. How Easy is Local Search? *Journal of Computer and System Sciences (JCSS)*, 37(1):79–100, 1988. ISSN 0022-0000. v, 7, 11, 12, 14, 19, 20, 21, 23, 31, 32, 33, 57, 135

[57] G. Kalai and D. Kleitman. A Quasi-Polynomial Bound for the Diameter of Graphs of Polyhedra. *Bulletin of the American Mathematical Society*, 26(2): 315–316, 1992. 9, 22

[58] M.-Y. Kao, editor. *Encyclopedia of Algorithms.* Springer, 2008. ISBN 978-0-387-30162-4. 6

[59] N. Karmarkar. A New Polynomial-Time Algorithm for Linear Programming. *Combinatorica*, 4(4):373–396, 1984. 9

[60] R. M. Karp. Reducibility Among Combinatorial Problems. In R. E. Miller and J. W. Thatcher, editors, *Complexity of Computer Computations*, The IBM Research Symposia Series, pages 85–103. Plenum Press, New York, USA, 1972. 5

[61] H. Kautz and B. Selman. The State of SAT. *Discrete Applied Mathematics*, 155(12):1514–1524, 2007. ISSN 0166-218X. 6

[62] L. Khachiyan. A Polynomial Algorithm in Linear Programming. *Doklady Akademiia Nauk SSSR*, 20(1):1093–1096, 1979. 9

[63] H. Klauck. On the Hardness of Global and Local Approximation. In *Proceedings of the 5th Scandinavian Workshop on Algorithm Theory (SWAT 1996)*, pages 88–99, London, UK, 1996. Springer-Verlag. ISBN 3-540-61422-2. 31, 34

[64] V. Klee and P. Kleinschmidt. The d-Step Conjecture and its Relatives. *Mathematics of Operations Research*, 12(4):718–755, 1987. 9

[65] V. Klee and G. J. Minty. How Good is the Simplex Algorithm? In O. Shisha, editor, *Inequalities III*, pages 159–175. Academic Press, 1972. 9

[66] Y. Kochetov and D. Ivanenko. Computationally Difficult Instances for the Uncapacitated Facility Location Problem. In T. Ibaraki, K. Nonobe, and M. Yagiura, editors, *Metaheuristics: Progress as Real Problem Solvers*, volume 32 of *Operations Research/Computer Science Interfaces Series*, pages 351–367. Springer US, 2005. ISBN 978-0-387-25383-1. 31

[67] M. W. Krentel. Structure in Locally Optimal Solutions (Extended Abstract). In *Proceedings of the 30th Annual Symposium on Foundations of Computer*

Science (FOCS 1989), pages 216–221. IEEE Computer Society, 1989. v, 16, 23, 32, 33, 41, 42, 55, 57, 65, 105, 109, 136, 137, 140

[68] M. W. Krentel. On Finding and Verifying Locally Optimal Solutions. *SIAM Journal on Computing (SICOMP)*, 19(4):742–749, 1990. ISSN 0097-5397. 33

[69] R. E. Ladner. The Circuit Value Problem is Log Space Complete for P. *SIGACT News*, 7:18–20, January 1975. ISSN 0163-5700. 58

[70] C. E. Lemke and J. J. T. Howsen. Equilibrium Points of Bimatrix Games. *SIAM Journal on Applied Mathematics*, 12(2):413–423, June 1964. 37

[71] S. Lin. Computer Solutions of the Traveling Salesman Problem. *Bell System Technical Journal*, 44:2245–2269, 1965. 8, 9, 32

[72] S. Lin and B. W. Kernighan. An Effective Heuristic Algorithm for the Traveling-Salesman Problem. *Operations Research*, 21(2):498–516, 1973. 9, 32

[73] G. S. Lueker. Manuscript. Princeton University, Princeton, NJ, 1975. 9

[74] M. Mavronicolas, I. Milchtaich, B. Monien, and K. Tiemann. Congestion Games with Player-Specific Constants. In L. Kucera and A. Kucera, editors, *Proceedings of the 32nd International Symposium on Mathematical Foundations of Computer Science (MFCS 2007)*, volume 4708 of *Lecture Notes in Computer Science*, pages 633–644. Springer, 2007. ISBN 978-3-540-74455-9. 36

[75] I. Milchtaich. Congestion Games with Player-Specific Payoff Functions. *Games and Economic Behavior*, 13(1):111–124, 1996. 36

[76] I. Milchtaich. The Equilibrium Existence Problem in Finite Network Congestion Games. In P. G. Spirakis, M. Mavronicolas, and S. C. Kontogiannis, editors, *Proceedings of the 2nd International Workshop on Internet and Network Economics (WINE 2006)*, volume 4286 of *Lecture Notes in Computer Science*, pages 87–98. Springer, 2006. ISBN 3-540-68138-8. 36

[77] B. Monien and T. Tscheuschner. On the Power of Nodes of Degree Four in the Local Max-Cut Problem. In T. Calamoneri and J. Díaz, editors, *Proceedings of the 7th International Conference on Algorithms and Complexity (CIAC 2010)*, volume 6078 of *Lecture Notes in Computer Science*, pages 264–275. Springer, 2010. ISBN 978-3-642-13072-4. 33

[78] B. Monien, D. Dumrauf, and T. Tscheuschner. Local Search: Simple, Successful, But Sometimes Sluggish. In Abramsky et al. [3], pages 1–17. ISBN 978-3-642-14164-5. 8, 17, 20, 22

[79] R. B. Myerson. *Game Theory: Analysis of Conflict.* Harvard University Press, September 1997. ISBN 0-674-34116-3. 13

[80] J. F. Nash. Non-Cooperative Games. *Annals of Mathematics*, 54(2):286–295, 1951. 12, 37

[81] N. Nisan, T. Roughgarden, E. Tardos, and V. V. Vazirani. *Algorithmic Game Theory*. Cambridge University Press, New York, NY, USA, 2007. ISBN 0521872820. 34

[82] J. B. Orlin, A. P. Punnen, and A. S. Schulz. Approximate Local Search in Combinatorial Optimization. In J. I. Munro, editor, *Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2004)*, pages 587–596. SIAM, 2004. 35

[83] P. N. Panagopoulou and P. G. Spirakis. Algorithms for Pure Nash Equilibria in Weighted Congestion Games. *J. Exp. Algorithmics*, 11:2.7, 2006. ISSN 1084-6654. 13

[84] C. Papadimitriou and K. Steiglitz. *Combinatorial Optimization – Algorithms and Complexity*. Prentice Hall, Englewood Cliffs, 1982. 6

[85] C. H. Papadimitriou. The Complexity of the Lin-Kernighan Heuristic for the Traveling Salesman Problem. *SIAM Journal on Computing (SICOMP)*, 21(3): 450–465, 1992. ISSN 0097-5397. 32

[86] C. H. Papadimitriou. On the Complexity of the Parity Argument and Other Inefficient Proofs of Existence. *Journal of Computer and System Sciences (JCSS)*, 48(3):498–532, 1994. 37

[87] C. H. Papadimitriou, A. A. Schäffer, and M. Yannakakis. On the Complexity of Local Search. In *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing (STOC 1990)*, pages 438–445, New York, NY, USA, 1990. ACM. ISBN 0-89791-361-2. 12, 20, 21, 22, 31, 32, 33

[88] V. T. Paschos. A Survey of Approximately Optimal Solutions to Some Covering and Packing Problems. *ACM Computing Surveys*, 29(2):171–209, 1997. 6

[89] S. Poljak. Integer Linear Programs and Local Search for Max-Cut. *SIAM Journal on Computing (SICOMP)*, 24(4):822–839, 1995. ISSN 0097-5397. 33

[90] O. A. Prokopyev, H.-X. Huang, and P. M. Pardalos. On Complexity of Unconstrained Hyperbolic 0-1 Programming Problems. *Operations Research Letters*, 33(3):312 – 318, 2005. ISSN 0167-6377. 31

[91] G. Reinelt. TSPLIB - A Traveling Salesman Problem Library. *INFORMS Journal on Computing*, 3(4):376–384, 1991. 9

[92] R. W. Rosenthal. A Class of Games Possessing Pure-Strategy Nash Equilibria. *International Journal of Game Theory*, 2:65–67, 1973. 13, 14, 24, 25, 51

[93] R. Savani and B. von Stengel. Hard-to-Solve Bimatrix Games. *Econometrica*, 74:397–429, 2006. 37

[94] T. J. Schaefer. The Complexity of Satisfiability Problems. In *Proceedings of the 10th Annual ACM Symposium on Theory of Computing (STOC 1978)*, pages 216–226. ACM, 1978. 33

[95] A. A. Schäffer and M. Yannakakis. Simple Local Search Problems that are Hard to Solve. *SIAM Journal on Computing (SICOMP)*, 20(1):56–87, 1991. ISSN 0097-5397. 12, 19, 21, 22, 31, 33, 42, 45, 108, 109

[96] L. S. Shapley. A Note on the Lemke-Howson Algorithm. In R. W. Cottle, L. C. W. Dixon, B. Korte, T. L. Magnanti, M. J. Todd, E. L. Allgower, R. Bartels, V. Chvatal, J. E. Dennis, B. C. Eaves, R. Fletcher, J.-B. Hiriart-Urruty, M. Iri, R. G. Jeroslow, D. S. Johnson, C. Lemarechal, L. Lovasz, L. McLinden, M. W. Padberg, M. J. D. Powell, W. R. Pulleyblank, K. Ritter, R. W. H. Sargent, D. F. Shanno, L. E. Trotter, H. Tuy, R. J. B. Wets, C. Witzgall, E. M. L. Beale, G. B. Dantzig, L. V. Kantorovich, T. C. Koopmans, A. W. Tucker, P. Wolfe, and M. L. Balinski, editors, *Pivoting and Extension*, volume 1 of *Mathematical Programming Studies*, pages 175–189. Springer Berlin Heidelberg, 1974. ISBN 978-3-642-00758-3. 10.1007/BFb0121248. 37

[97] S. Shimozono. Finding Optimal Subgraphs by Local Search. *Theoretical Computer Science*, 172(1-2):265 – 271, 1997. ISSN 0304-3975. 34

[98] A. Skopalik and B. Vöcking. Inapproximability of Pure Nash Equilibria. In R. E. Ladner and C. Dwork, editors, *Proceedings of the 40th Annual ACM Symposium on Theory of Computing (STOC 2008)*, pages 355–364. ACM, 2008. ISBN 978-1-60558-047-0. 36, 108, 139

[99] S. Smale. On the Average Number of Steps in the Simplex Method of Linear Programming. *Mathematical Programming*, 27:241–262, 1983. 10

[100] R. Solis-Oba. Local Search. In T. F. Gonzalez, editor, *Handbook of Approximation Algorithms and Metaheuristics (Chapman & Hall/CRC Computer & Information Science Series)*. Chapman & Hall/CRC, 2007. 7

[101] D. A. Spielman and S.-H. Teng. Smoothed Analysis of Algorithms: Why the Simplex Algorithm Usually Takes Polynomial Time. *Journal of the ACM (JACM)*, 51(3):385–463, 2004. 10

[102] L. J. Stockmeyer and A. R. Meyer. Word Problems Requiring Exponential Time: Preliminary Report. In *Proceedings of the 5th Annual ACM Symposium on Theory of Computing (STOC 1973)*, pages 1–9. ACM, 1973. 5

[103] M. Todd. The Many Facets of Linear Programming. *Mathematical Programming*, 91(3):417–436, 2001. 7, 9

[104] T. Tscheuschner. Settling the Complexity of Local Max-Cut (Almost) Completely. *The Computing Research Repository (CoRR)*, abs/1004.5329, 2010. 15, 33, 34, 108, 109, 140

[105] A. Vattani. k-Means Requires Exponentially Many Iterations Even in the Plane. In J. Hershberger and E. Fogel, editors, *Proceedings of the 25th ACM Symposium on Computational Geometry (SoCG 2009)*, pages 324–332. ACM, 2009. ISBN 978-1-60558-501-7. 10

[106] B. von Stengel. Computing Equilibria for Two-Person Games. In R. Aumann and S. Hart, editors, *Handbook of Game Theory with Economic Applications*, volume 3 of *Handbook of Game Theory with Economic Applications*, chapter 45, pages 1723–1759. Elsevier, May 2002. 38

[107] T. Vredeveld and J. K. Lenstra. On Local Search for the Generalized Graph Coloring Problem. *Operations Research Letters*, 31(1):28 – 34, 2003. ISSN 0167-6377. 31, 33

[108] M. Yannakakis. The Analysis of Local Search Problems and Their Heuristics. In C. Choffrut and T. Lengauer, editors, *Proceedings of the 7th Annual Symposium on Theoretical Aspects of Computer Science (STACS 1990)*, volume 415 of *Lecture Notes in Computer Science*, pages 298–311. Springer, 1990. ISBN 3-540-52282-4. 7

[109] M. Yannakakis. Computational Complexity. In E. Aarts and J. Lenstra, editors, *Local Search in Combinatorial Optimization*, pages 19–55. Wiley, Chichester, 1997. 12, 20, 21, 31

[110] M. Yannakakis. Equilibria, Fixed Points, and Complexity Classes. *Computer Science Review*, 3(2):71–85, 2009. ISSN 1574-0137. 7, 38