Bachelorarbeit am Institut für Informatik der Freien Universität Berlin,

Arbeitsgruppe Theoretische Informatik

The complexity class Polynomial Local Search (PLS) and PLS-complete problems

Michaela Borzechowski

Matrikelnummer: 4677938

michaela.borzechowski@fu-berlin.de

Erstgutachter: Prof. Dr. Wolfgang Mulzer
Zweitgutachter: M.Sc. Yannik Stein

Berlin, September 19, 2016

Eigenständigkeitserklärung

Ich erkläre hiermit, dass ich die vorliegende Bachelorarbeit selbstständig und ohne unerlaubte Hilfe angefertigt habe. Alle verwendeten Quellen und Hilfsmittel sind im Literaturverzeichnis aufgeführt und wörtlich oder inhaltlich aus den benutzten Quellen entnommene Stellen sind als solche kenntlich gemacht. Ich erkläre weiterhin, dass diese Arbeit nicht im Rahmen eines anderen Prüfungsverfahrens eingereicht wurde.

Michaela Borzechowski, Berlin, September 19, 2016

Abstract

The complexity classes P and NP are well known. However we are often interested in the actual globally optimal solutions of some NP decision problems. Local search is an attempt to approximate a hard to find global optimum with a local optimum. The complexity class Polynomial Local Search (PLS) was introduced to analyze the complexity of local search algorithms, where it is verifiable in polynomial time, whether a solution is a local optimum or not. One can PLS-reduce local search problems to one another and establish PLS-completeness. This work presents the basic definitions of the class PLS, its relation to other complexity classes, PLS-reductions, PLS-completeness, as well as a list of PLS-complete problems. The aim is to give a general overview of this topic and make further proofs for PLS-completeness and further investigations of the characteristics of the class PLS easier.

Contents

1. Introduction and Motivation

The complexity classes P and NP are well known. Informally, P is the class containing decision problems that are solvable in polynomial time, and NP consists of the decision problems "verifiable" in polynomial time [Cor+09, p.1049]. However we are interested in the actual globally optimal solutions of some NP problems, and not just the answer to the decision problem. For example in the *Traveling Salesman Problem* we want to know which tour is the shortest tour. There are local search heuristics as an attempt to solve optimization versions of problems in NP, for example Lin-Kernighan for the *Traveling Salesmen Problem*. They try to approximate the global optimum with a local optimum, for example with an iterative improvement algorithm [LK73].

When searching for a local optimum, there are two interesting issues to deal with: First how to find a local optimum, and second how long it takes to find a local optimum.

For many of these local search algorithms, we do not know, whether they can find a local optimum in polynomial time or not [Yan88]. So to answer the question of how long it takes to find a local optimum, Johnson, Papadimitriou and Yannakakis [JPY88] introduced the complexity class PLS in their paper "How easy is local search". It contains local search problems for which the local optimality can be verified in polynomial time. It lies somewhere between P and NP, more formally this will be explained in Section 2.2 on page 6. They introduce a PLS-reduction, in order to reduce one local search problem to one another, and to establish PLS-complete problems. If a polynomial time algorithm is found for a PLS-complete problem, all PLS-complete problems can be solved in polynomial time. Furthermore, proving a problem to be PLS-complete, proves that if the problem is NP-hard, then NP=co-NP.

After Johnson, Papadimitriou and Yannakakis first introduced the class PLS in the Paper [JPY88] in 1988, Schäffer and Yannakakis proved several well known problems like *Max-Cut* and *Stable configuration*, to be PLS-complete in [SY91]. More recently *Congestion games* were proved to be PLS-complete in [FPT04].

Though several problems are proven to be PLS-complete, the knowledge about this class is still very limited. In the following, a formal definition of PLS will be presented, its relation to NP and P, or rather the optimization versions of NP and P are explained, a first PLS-complete problem is shown and a list of other PLS-complete problems is presented.

The aim of this work is to give a general overview of this topic and to show for a few examples how proofs of PLS-completeness proceed. This shall make it easier for researchers in the future to prove further problems to be PLS-complete and to further investigate the characteristics of the class PLS.

2. Basics

2.1. What is local search?

Let $\Sigma := \{0, 1\}$. We can view a decision problem $D : \Sigma^* \to \{0, 1\}$ as the language $Z = \{x \in \Sigma^* : D(x) = 1\}$. [Cor+09, p.1058]

A language Z is in P if there exists a deterministic Turing machine that decides in polynomial time whether or not $x \in \Sigma^*$ is in Z. [Cor+09, p.1059]

Z is in NP if and only if there exits a deterministic Turing machine A and a constant $c \in \mathbb{R}^+$ so that $Z = \{x \in \Sigma^* :$ there exists a certificate $y \in \Sigma^*$ with $\mid y \mid = O(\mid x \mid^c)$ such that $A(x, y) = 1\}$.

We say that algorithm A verifies language Z in polynomial time. [Cor+09, p.1064]

NP contains decision problems, but many problems of interest are optimization problems, so problems where we search for an optimal solution.

More mathematically, we can view a search problem Q as relation $R \subseteq \Sigma^* \times \Sigma^*$, where a pair $(I, s) \in R$ represents an input instance I and a searched solution s. An algorithm "solves" R, if given an instance $I \in \Sigma^*$, it finds a solution $s \in \Sigma^*$ so that $(I, s) \in R$, or it states correctly that no such s exists. [JPY88, p. 84]

Combinatorial global optimization problems are a special case of search problems where we want to find a solution that maximizes or minimizes a cost function.

Definition 2.1 Combinatorial global optimization problem

A problem OP is a combinatorial global optimization problem if it is a search problem with a set of problem instances $D_{OP} \subseteq \Sigma^*$ where $I \in D_{OP}$ is a particular problem instance. $F_{OP}(I)$ is the finite set of solutions $s \in \Sigma^*$ for an instance I. A cost function $c_{OP} : (I, s) \mapsto x$, where $x \in \mathbb{R}$, assigns a cost to each solution s of an instance I. The aim is to find a global optimal solution for each instance I, which has the best cost of all solutions. If the problem is a minimization problem, the global optimal solution $s^* \in F_{OP}(I)$ is a solution with the lowest cost, so $c_{OP}(I, s^*) \leq c_{OP}(I, s)\ \forall s \in F_{OP}(I)$, if the problem is a maximization problem it is the solution with the highest cost, so $c_{OP}(I, s^*) \geq c_{OP}(I, s)\ \forall s \in F_{OP}(I)$. [Cor+09, p.1050], [MAK07, Definition 1.1 and Definition 1.2]

Corresponding to P and NP we define FP and FNP as complexity classes for search problems.

Definition 2.2 FNP

A search problem R is in FNP if

- There is a polynomial time algorithm V that, given a pair (I, s), determines in polynomial time whether or not (I, s) is in R
- If $(I, s) \in R$, then $\mid s \mid$ is polynomially bounded in $\mid I \mid$

[Yan88, p.28], [MP91]

In other words, if a combinatorial global optimization problem is in FNP, then there is a nondeterministic polynomial time algorithm that solves it. [Yan88, p.28]

Definition 2.3 TFNP

TFNP is the subset of problems of FNP, where there always exists a solution for the given problem. [MP91]

Definition 2.4 FP

A search problem R is in FP if it is in FNP and if there exists a deterministic polynomial time algorithm that solves it. So given an instance I, it returns a solution s so that $(I, s) \in R$ or it states correctly that such an s does not exist, in polynomial time. [Yan88, p.28], [MP91]

If a combinatorial global optimization problem is in FP, then there is a deterministic polynomial time algorithm that solves it. [Yan88, p.28]

Lemma 2.1 *FP=FNP if and only if P=NP.*

Proof. It is shown in [JPY88] that FP=FNP if and only if P=NP.

A local search problem L is similar to a combinatorial global optimization problem OP, with the difference, that it has aditionally a so called neighborhood structure. A solution s has one or more neighboring solutions within this neighborhood structure.

Definition 2.5 Local search problem

A local search problem L is a combinatorial global optimization problem with an additional neighborhood structure. A problem L is a local search problem, if it has the same problem instances $D_L = D_{OP}$ as OP, where $I \in \Sigma^*$ is one problem instance of the set D_L. Furthermore it has the same finite set of solutions $F_L(I) = F_{OP}(I)$ for each instance $I \in D_L$ where $s \in F_L(I)$ is a particular solution. The cost function is $c_L(I, s) = c_{OP}(I, s)$. The difference between the local search and the combinatorial global optimization problem is that the local search problem has additionally a neighborhood structure $N : (I, s) \mapsto f$, where $f \subseteq F_L(I)$.

Table 1: Overview of the above explained terms and symbols

L	A local search problem
D_L	$D_L \subseteq \Sigma^*$, all instances of the problem L
I	$I \in D_L$ one particular problem instance I of L
$F_L(I)$	Set of all solutions for instance I of problem L
s	$s \in F_L(I)$, one solution of an instance I
$c_L(I, s)$	$c_L : (I, s) \mapsto r, r \in \mathbb{R}^+$, Non-negative cost of solution s for an instance I of problem L
$N(I, s)$	$N : (I, s) \mapsto f, f \subseteq F_L(I)$, set of neighbors of solution s for an instance I of problem L

Definition 2.6 Local optimum

The goal of a local search problem is to find a local optimum, which is a solution s, that has no neighbor with better costs. So if L is a minimization problem and s a local optimum, no neighbors of s have lower costs, and if it is a maximization problem, no neighbor of s has higher costs than s itself.

The associated local search problem of an optimization problem cannot be computationally harder to compute than the original global optimization problem, as a global optimum is always a local optimum too. [MAK07]

Example 2.1 Let the problem L be Max-2SAT (for a formal definition see Section 4.2 on page 35). The instances D_L of the problem are boolean formulas in conjunctive normal form, with at most two literals in each clause.
Consider the following instance I: $(x_1 \vee x_2) \wedge (\neg x_1 \vee x_3) \wedge (\neg x_2 \vee x_3)$.
A solution s for that instance is a bit string that assigns every x_i the value 0 or 1. In this case, a solution consists of 3 bits, for example $s = 000$, which stands for the assignment of x_1 to x_3 with the value 0. The set of solutions $F_L(I)$ is the set of all possible assignments for x_1, x_2 and x_3. The cost of each solution is the number of satisfied clauses, so $c_L(I, s = 000) = 2$ because the second and third clause are satisfied. The neighbor of a solution s is reached by flipping one bit of the bitstring s, so the neighbors of s are $N(I, 000) = \{100, 010, 001\}$ with the following costs:
$c_L(I, 100) = 2$
$c_L(I, 010) = 2$
$c_L(I, 001) = 2$
There are no neighbors with better costs than s, if we are looking for a solution with maximum cost. Even though s is not a global optimum (which for example would be a solution $s' = 111$ that satisfies all clauses and has $c_L(I, s') = 3$), s is a local optimum, because none of its neighbors has better costs.

Note that the neighborhood structure does not have to be symmetric. If a solution $s \in F_L(I)$ has a neighbor $r \in F_L(I)$ then s does not need to be a neighbor of r.

The neighborhood structure in the example above was the Flip structure, which is a simple structure. There are others and more sophisticated structures such as the Kernighan-Lin neighborhood structure, where a neighbor r of s is obtained by a sequence of "greedy" flips and none of the bits once flipped are allowed to be flipped again. This neighborhood is used for the traveling salesman problem in Section 4.10 on page 45.

Definition 2.7 Exact neighborhood

A neighborhood where every local optimum is a global optimum too is called an exact neighborhood. [Yan88], [MAK07, Definition 1.10]

For example the neighborhood structure that has every solution as neighbors of each solution is an exact neighborhood structure, though it is very inefficient to test for a problem with an exponential solution space whether or not a solution is a local optimum. A local search problem with an exact neighborhood is equivalent to the combinatorial global optimization problem.

An example of an exact neighborhood can be found in linear programming, where the solutions are the vertices of a polytope and the neighborhood structure is given by the edges of the polytope. In this case a local optimum is a global optimum. Maximum matching is another example of an exact neighborhood, where a matching r is the neighbor of the matching s if their symmetric difference is an augmenting path. Furthermore minimum spanning tree is an example of an exact neighborhood too, where a tree r is neighbor of a tree s, if it can be obtained by adding an edge and removing an edge from the unique cycle that is thus formed. [Yan88, p.24]

2.2. The class PLS

We define a complexity class called PLS to capture the local search problems with the characteristic that we can search the neighborhood of a solution in polynomial time. Therefore we are able to verify whether or not a solution is a local optimum in polynomial time.

Definition 2.8 The complexity class PLS (Formal)

Let L be a local search problem and R the relation that models L. The relation

$$R \subseteq D_L \times F_L(I) := \{(I, s) \mid I \in D_L, s \in F_L(I)\}$$

is in PLS if

- The size of every solution $s \in F_L(I)$ is polynomial bounded in the size of I
- Problem instances $I \in D_L$ and solutions $s \in F_L(I)$ are polynomial time verifiable
- There is a polynomial time computable function $A : D_L \to F_L(I)$ that returns for each instance $I \in D_L$ some solution $s \in F_L(I)$
- There is a polynomial time computable function $B : D_L \times F_L(I) \to \mathbb{R}^+$ that returns for each solution $s \in F_L(I)$ of an instance $I \in D_L$ the cost $c_L(I, s)$
- There is a polynomial time computable function $N : D_L \times F_L(I) \to Powerset(F_L(I))$ that returns the set of neighbors for an instance-solution pair
- There is a polynomial time computable function $C : D_L \times F_L(I) \to N(I, s)$ that returns a neighboring solution s' with better cost than solution s, or states that s is locally optimal
- For every instance $I \in D_L$, R exactly contains the pairs (I, s) where s is a local optimal solution of I

[MS14]

To simplify matters, we will say a problem L is in PLS instead of saying the relation R that models the problem L is in PLS.

Table 2: L is in PLS if there exist the following three polynomial time computable functions A, B and C

$A(I)$	Returns some solution $s \in F_L(I)$ for the problem instance I
$B(I, s)$	Determines whether $s \in F_L(I)$ and if so, computes the cost of solution s
$C(I, s)$	If there exists a neighbor s' of solution s with cost $c_L(s')$ computed by B better than $c_L(s)$, C returns s', otherwise it states that s is a local optimum

One can use these algorithms for example like it is done in the Standard Algorithm (Algorithm 1 on page 10).

We can now compare PLS with FP and FNP, to determine what the complexity of PLS is.

Lemma 2.2 *FP $\subseteq$ PLS $\subseteq$ FNP*

Proof. FP $\subseteq$ PLS

Let $Q \in$ FP be a minimization problem, then there exists by definition an algorithm M_Q that solves Q in polynomial time.
We can build the polynomial time algorithms A, B and C, so that Q matches a PLS problem L_Q.
Let $A(I)$ return the solution M_Q returns. As M_Q runs in polynomial time, so does A.
Let $B(I, s)$ return 0 if s is the global optimum M_Q returns, and 1 if not.
The neighborhood that is needed for the PLS problem is defined as follows: The neighbor of each solution is the solution M_Q returns. Therefore let $C(I, s)$ return the solution M_Q returns.
Therefore L_Q is in PLS.

Proof. PLS $\subseteq$ FNP

Let L be a problem in PLS. By definition there exist the polynomial time algorithms A, B and C.
The length of a solution is polynomial bounded in the length of the input by definition.
We can construct a polynomial time algorithm V, that determines whether or not an instance-solution pair (I, s) is in R, which means that s is a local optimal solution of I.
V uses algorithm $C(I, s)$ that says in polynomial time whether or not s is a locally optimal solution of I or not.
Therefore L is in FNP. [JPY88], [Yan88]

Lemma 2.3 *PLS $\subseteq$ TFNP*

Proof. A PLS problem has always a local optimum and therefore always a solution, as the set of solutions is finite. It follows that it is a subset of TFNP. [MP91, p.319]

Lemma 2.4 *If a PLS problem is NP-hard, then NP = co-NP.*

Proof. The following is meant by "a PLS problem is NP-hard": A problem L is NP-hard, if one can transform an instance of an NP problem X in polynomial time to an instance of L, then solve L instead of X, and transform the solution of L back to X. One can imagine this as building a deterministic Turing machine that builds an Instance I_L out of I_X, asks an oracle for L once or polynomial times often for an answer, and then concludes from these answers whether I_X is in the language of X or not (Figure 1 on the following page). If L was solvable in polynomial time, X would be solvable in polynomial time too.

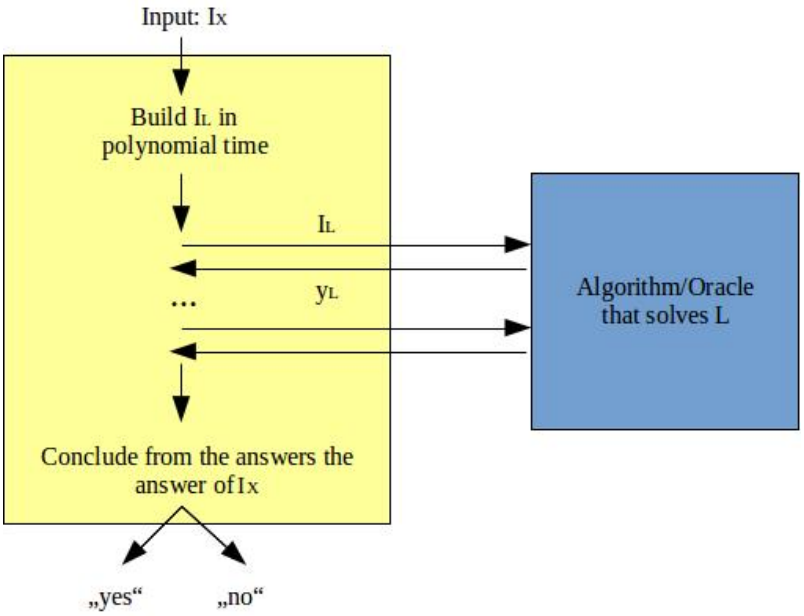


Figure 1: The Oracle Turing Machine

Now we show, that if a PLS problem L is NP-hard, we can verify a "no" instance, so a not solvable instance of an NP-complete problem X, by building the following non-deterministic Turing machine.

Let L be a PLS problem that is NP-hard. Let I_X be an instance of a NP-complete problem X that has no solution. We can transform I_X in polynomial time to an instance I_L of L. Now, instead of asking an oracle, we guess a solution for I_L. There is always a local optimum for every instance of L, as L is a PLS problem.

After guessing a solution, we verify whether we guessed right or not, so whether the guessed solution y_L is in L or not. This can be done in polynomial time by the algorithm C_L, which exists by the definition of a PLS problem.

If y_L is not a solution for L, we simply return "no".

If y_L is a solution for L, we continue as before: We conclude that I_X is either a "yes, there is a solution for decision problem X" or a "no, there is no solution for decision problem X" instance. We swap the answers, so that if there is "no" solution for decision problem X, we return a "yes", and a "no", if there exists a solution for X.

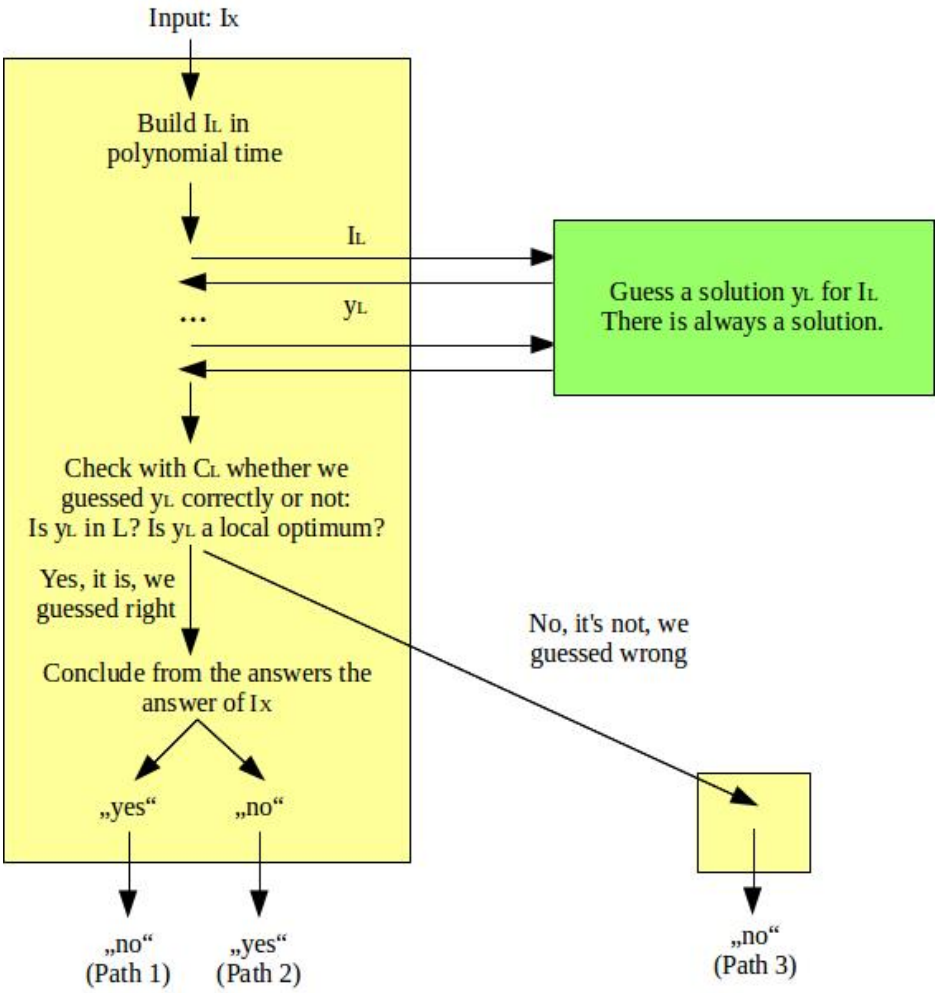


Figure 2: Nondeterministic Turing Machine that verifies that X is in co-NP

If I_X is a solvable instance, "no" is always returned: Either we guess a not locally optimal solution, then a "no" is returned (Path 3), or we guess a local optimum, then we return "no" too (Path 1), as the machine concludes that there is a solution for I_X. If I_X is a not-solvable instance, there exists a path to "yes" (Path 2) and as we guess non-deterministically the answer needed, we can reach the "yes".

Therefore we can now verify with a non-deterministic Turing machine whether X is in co-NP or not. As X is NP-complete, this implies that NP=co-NP. [JPY88]

2.3. The standard Algorithm

Definition 2.9 Standard algorithm

One way to solve each local search problem is through iterative improvement. The following algorithm is called standard algorithm [JPY88]. It starts with an initial solution $s = A(I)$ and then iteratively searches for a better neighbor of s:

$s = A(I)$; `// Start with an initial solution`
$s' = C(I, s)$; `// Calculate a better neighbor of` s
while *s is not a local optimum, $B(I, s')$ is better than $B(I, s)$* **do**
    $s = s'$;
    $s' = C(I, s)$;
end
return s ; `// If there is no better neighbor, we found a local optimum`

Algorithm 1: The Standard Algorithm

Note that it is not necessary always to use the standard algorithm, every algorithm is permitted. For example the standard local search algorithm for Linear Programming is the Simplex algorithm, where the solutions are the vertices of a polytope and the neighborhood structure is given by the edges of the polytope.

But even though all three functions A, B and C run in polynomial time, this does not necessarily mean that the whole standard algorithm runs in polynomial time. If the solutions only have a polynomial number of possible different costs, the standard algorithm runs in polynomial time. In each iteration we improve the costs and if the costs are polynomial bounded, the number of iterations is polynomial bounded too. If the number of different possible costs for a solution can be exponential, the standard algorithm needs in the worst case an exponential number of iterations too. Therefore the standard algorithm is pseudo polynomial in the number of different costs of a solution. [ACZ14, p.438]

The space the standard algorithm needs is only polynomial. It only needs to save the current solution s, which is polynomial bounded by definition. [Yan88, p.47]

Finding one particular local optimum with the standard algorithm is harder than just finding any local optimum. This problem is called the standard local optimum problem and it is PSPACE-complete. [Yan88, p.47]

2.4. PLS-reduction

Definition 2.10 PLS-reduction

A local search problem L_1 is PLS-reducable to a local search problem L_2 if there are two polynomial time functions $f : D_1 \to D_2$ and $g : D_1 \times F_2(f(I_1)) \to F_1(I_1)$ such that:

- if I_1 is an instance of L_1, then $f(I_1)$ is an instance of L_2
- if s_2 is a solution for $f(I_1)$ of L_2, then $g(I_1, s_2)$ is a solution for I_1 of L_1
- if s_2 is a local optimum for instance $f(I_1)$ of L_2, then $g(I_1, s_2)$ has to be a local optimum for instance I_1 of L_1

(f, g) is a PLS-reduction and we write $L_1 \preceq L_2$ if L_1 is PLS-reducable to L_2.

So in order to actually reduce a problem L_1 to a problem L_2, the function f has to be defined, so that it transforms the instances of L_1 to instances of L_2 and g has to be defined so that the solutions of L_2 are mapped to the solutions of L_1. Furthermore it must be proven that a local optimum of L_2 is a local optimum of L_1 too. There are examples in Section 4 on page 21.

It is sufficient to only map the local optima of $f(I_1)$ to the local optima of I_1, and to map all other solutions for example to the standard solution returned by A_1. [MAK07]

It is easy to see, that the PLS-reduction is transitive, that means if a problem L_1 can be reduced to a problem L_2, and L_2 can be reduced to a problem L_3, then L_1 can be reduced to L_3 [JPY88], [Yan88]. We can simply take the instance $I_2 = f_1(I_1)$ and continue building $I_3 = f_2(f_1(I_1))$. The same holds for g: If s_3 is local optimum of I_3, then $s_2 = g_2(I_2, s_3)$ is local optimum of I_2 and $s_1 = g_1(I_1, g_2(I_2, s_3))$ is local optimum of I_1.

2.5. Tight PLS-reduction

Definition 2.11 Transition Graph

The transition graph T_I of an instance I of a problem L is a directed graph. The nodes represent all elements of the finit set of solutions $F_L(I)$ and the edges point from one solution to the neighbor with strictly better cost, so it is an acyclic graph. A sink, which is a node with no outgoing edges, is a local optimum. The height of a vertex v is the length of the shortest path from v to the nearest sink. The height of the transition graph is the largest of the heights of all vertices, so it is the height of the largest shortest possible path from a node to its nearest sink. [MAK07, height = potential p.7]

The standard local search algorithm moves from node to node until it reaches such a sink. The number of iterations the standard algorithm needs to find a local optimum is the number of nodes on the path from the starting solution to the sink. The height of the transition graph gives a lower bound on the number of iterations the standard algorithm needs at most, no matter which path it takes, if there are two or more equally best neighbors. [MAK07, p.7]

It follows Lemma 2.5.

Lemma 2.5 *If a PLS problem has an instance for which its transition graph has an exponential height, then the standard local search algorithm needs exponential time in the worst case, independent of the way in which it chooses the best neighboring solution. [ACZ14]*

There exists a problem that is in PLS and has an exponential height.

Example 2.2 Let L be a minimization problem that takes an integer n and asks for the smallest integer between 0 and 2^n. Let the cost function be $c(I, s) = s$ with $0 \leq s \leq 2^n$. The neighbor of a solution s for $s \geq 1$ has only one neighbor, $N(I, s) = \{s - 1\}$. The local and at the same time the global optimum of this problem is obviously 0. The transition graph is a chain from node 2^n to node 0, therefore the standard algorithm for this problem needs an exponential number of iterations. [ACZ14, p.449]

The more interesting question is, if we have one PLS problem L_1 where the standard algorithm needs an exponential number of iterations, and we reduce L_1 to L_2: Does the standard algorithm needs an exponential number of iterations for L_2 too?

If we just use the normal PLS-reduction, we cannot prove this, so we define tight PLS-reduction, for which the answer to this question is yes.

Definition 2.12 Tight PLS-reduction

A PLS-reduction (f, g) from a local search problem L_1 to a local search problem L_2 is a tight PLS-reduction if for any instance I_1 of L_1 we can choose a subset $\mathscr{R}$ of solutions of instance $I_2 = f(I_1)$ of L_2, so that the following properties are satisfied:

- $\mathscr{R}$ contains, among other solutions, all local optima of I_2
- For every solution p of I_1, we can construct in polynomial time a solution $q \in \mathscr{R}$ of $I_2 = f(I_1)$ so that $g(I_1, q) = p$
- If the transition graph $T_{f(I_1)}$ of $f(I_1)$ contains a direct path from q to q', and q, $q' \in \mathscr{R}$, but all internal path vertices are outside $\mathscr{R}$, then for the corresponding solutions $p = g(I_1, q)$ and $p' = g(I_1, q')$ holds either $p = p'$ or T_{I_1} contains an edge from p to p'

[SY91]

Lemma 2.6 *If a PLS problem L_1 is tight-PLS-reducable to a PLS problem L_2, then the following holds: The height of $T_{f(I_1)}$ is at least as large as the height of T_{I_1}. So if the standard algorithm that solves L_1 takes exponential time in the worst case, the standard algorithm that solves L_2 takes exponential time in the worst case too. [SY91, Lemma 3.3]*

Proof. Let I_1 be an instance of L_1 and T_{I_1} its transition graph. Let p be a solution node, which has the same height as T_{I_1}, by definition there must be such a node, as the height of the graph is the biggest height of the vertices. Let $I_2 = f(I_1)$ and $T_{f(I_1)}$ be the corresponding transition graph. Let $q \in \mathscr{R}$ be a solution for I_2, with $g(I_1, q) = p$. The height of q in $T_{f(I_1)}$ is at least as large as the height of p in T_{I_1}: Consider the shortest path from q to a sink in $T_{f(I_1)}$. Let the vertices on the path from q to the sink that are in $\mathscr{R}$ be q_1, q_2, ... , q_k. Let be p_1, p_2, ... , p_k the images of the corresponding q_i's. By the definition of the tight reduction we know that q_k is a local optimum of $f(I_1)$, and therefore p_k is a local optimum of I_1. Because of the third point of the definition of a tight reduction we know, that if there is a path from q_i to q_{i+1}, then there is an edge from p_i to p_{i+1} or $p_i = p_{i+1}$, as all q_i are in $\mathscr{R}$, and the vertices between them are not. So for each i either $p_i = p_{i+1}$ or there is an arc in T_{I_1} from p_i to p_{i+1}. Therefore there is a path of length at most k from vertex p to a sink of T_{I_1}. [SY91, Proof of Lemma 3.3]

2.6. PLS-completeness

Definition 2.13 PLS-completeness

A local search problem L is PLS-complete, if

- L is in PLS
- every problem in PLS can be PLS-reduced to L

We can show that we can reduce every PLS problem to the problem Circuit/Flip, which is in PLS, and therefore is PLS-complete (See Theorem 3.1 on page 16). If we want to prove another PLS problem L to be PLS-complete, it is sufficient to show that there is a reduction from an PLS-complete problem (like Circuit/Flip) to L, which is much easier than reducing every PLS problem to L.

PLS-completeness implies that if there is a polynomial time algorithm that finds a local optimum for one PLS-complete problem, then there is a polynomial time algorithm for every PLS-complete problem. Until now there is no such polynomial time algorithm found. If there is a polynomial time algorithm, it follows that FP=PLS. [MAK07, p.127]

All in all, proving a problem L to be PLS-complete tells us that a local optimum can be verified in polynomial time, and that it is very unlikely that L is NP-hard or solvable in polynomial time.

Definition 2.14 Tight PLS-completeness

A problem L is tight PLS-complete, if L is in PLS and if each problem in PLS is tightly PLS-reducable to L.

Knowing that the height of the transition graph can be exponentially large for a given PLS-complete problem, proves that the worst case running time of iterative improvement is exponential, even if FP=PLS. [MAK07, p.127]

3. A first PLS-complete problem: Circuit/Flip

The first PLS-complete problem will be Circuit/Flip. It can be either a minimization or a maximization problem.

Definition 3.1 Circuit/Flip

An instance I of Circuit/Flip is an acyclic boolean circuit with x_1, ..., x_n inputs and y_1, ..., y_m outputs. It can be imagined for example as an actual circuit or a $m \times 1$ vector of boolean expressions.
A solution s of I is a certain assignment of x_1, ..., x_n.
The cost of a solution x_1, ..., x_n is the output y_1, ..., y_m read as an integer number, so:

$$c(x_1, ..., x_n) = \sum_{i=1}^{m} 2^{(i-1)}(y_i)$$

The neighborhood of a solution $s = x_1$, ..., x_n can be achieved by negating (flipping) one arbitrary input bit x_i. So one solution s and all its neighbors $r \in N(I, s)$ have Hamming distance one: $H(s, r) = 1$.

Max-circuit/Flip searches for a solution with maximum cost and Min-circuit/Flip for a solution with minimum cost.

Example 3.1 Max-circuit/Flip

$$\begin{pmatrix} x_1 \vee x_2 \\ (x_1 \vee x_2) \wedge \neg x_3 \end{pmatrix}$$

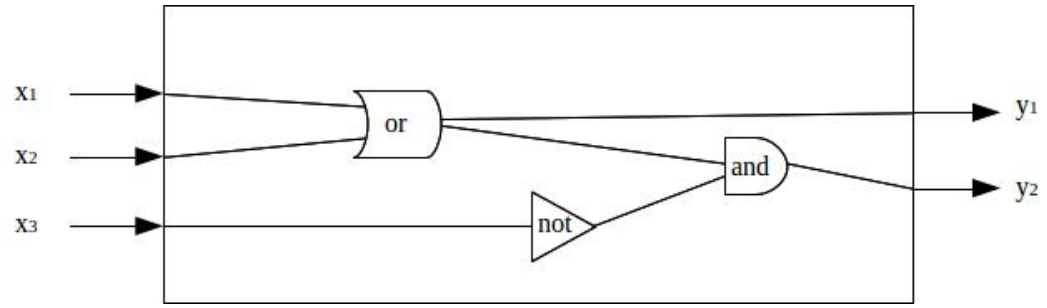


Figure 3: Circuit I

Table 3: Solutions and costs

One solution s:	$x_1 = 1$, $x_2 = 1$, $x_3 = 1$
Output of solution s:	$y_1 = 1$, $y_2 = 0$
Cost of solution s:	as binary: 01, as decimal: 1
Neighbors of solution s:	$x_1 = 1$, $x_2 = 1$, $x_3 = 0$ with output $y_1 = 1$, $y_2 = 1$ and cost in decimal 3
	$x_1 = 1$, $x_2 = 0$, $x_3 = 1$ with output $y_1 = 1$, $y_2 = 0$ and cost in decimal 1
	$x_1 = 0$, $x_2 = 1$, $x_3 = 1$ with output $y_1 = 1$, $y_2 = 0$ and cost in decimal 1
Local optimum:	$x_1 = 1$, $x_2 = 1$, $x_3 = 0$ with output $y_1 = 1$, $y_2 = 1$ and cost in decimal 3

Lemma 3.1 *Max-circuit/Flip and Min-circuit/Flip are equivalent.*

Proof. Max-circuit/Flip and Min-circuit/Flip can be reduced to each other. This can be achieved by adding an additional layer of logic to the circuit that negates every output. Let g be the identity and define f to transform the instances by adding a *not*-gate in front of every output variable y_i, this can be done in polynomial time. Obviously the local and global optima are the same.

Theorem 3.1 *Min-circuit/Flip and Max-circuit/Flip are PLS-complete.*

Proof. Since Min-circuit/Flip and Max-circuit/Flip are equivalent, we will show in the following PLS-completeness for Min-circuit/Flip, which proves Max-circuit/Flip to be PLS-complete too.

In order to prove PLS-completeness, we need to show that Min-circuit/Flip is in PLS. It obviously is, as there exist all three algorithms A, B and C that run in polynomial time:
$A(I)$ produces some solution, for example the word where every x_i is assigned to 1. As the length of any solution is polynomial bounded in the number of input bits, A runs in polynomial time.
$B(I, s)$ calculates the cost of a solution, which can be done in polynomial time too.
$C(I, s)$ searches the neighborhood for a better neighbor. The set $N(I, s)$ has size n, as it contains each solution with exactly one of the n input bits flipped. Therefore, C can be computed in polynomial time too.

Furthermore we need to reduce every problem in PLS to Min-circuit/Flip.

Let L be any fixed PLS problem with a polynomial function p that bounds the size of the solutions to the size of the instance I. A solution has length $p(|I|)$. The Hamming distance of all solutions $s \in F_L(I)$ is greater than 1. This property can be ensured by dublicating each bit in the encoding of a solution [MAK07, p.105]. Without loss of generality let L be a minimization problem.

Let Q be another fixed PLS problem. It has the same instances as L, but a different neighborhood structure: A solution r is neighbor of a solution s if and only if their Hamming distance is exactly 1.

We prove that $L \preceq Q$ and $Q \preceq$ Min-circuit/Flip, which proves that every PLS problem can be reduced to Min-circuit/Flip.

We need Q in between in order to achieve the neighborhood structure of Hamming distance one between two neighbors, which is required for the Flip neighborhood.

Table 4: Overview

	L	Q	Min-circuit/Flip
Instance	I	I	$f(I)$
If r is neighbor of s then	$H(s,r) > 1$	$H(s,r) = 1$	$H(s,r) = 1$
Length of solution	$p(\mid I \mid)$	$2p(\mid I \mid) + 2 = 2p + 2$	$m \in \mathbb{N}$

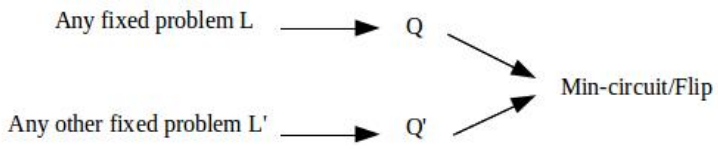


Figure 4: Reductions

Lemma 3.2 $L \preceq Q$

Proof. Let u be a solution of L with length $p(|I|)$. The length of solutions q of Q is defined as $2p(|I|) + 2$. In order to reduce L to Q, f and g need to be defined.
Let f be the identity, as L and Q have the same problem instances.
g encodes the solutions of Q in the following way:

Let $\mathscr{R}$ be the subset of solutions of Q that have the structure $r = uu00$ where u is solution of L encoded as binary: $\mathscr{R} := \{uu00 | u \in F_L(I), u \text{ is solution of } L \text{ encoded as binary}\}$.

Then let g be defined as a function that returns u if $r \in F_Q(I)$ is in $\mathscr{R}$ and otherwise the standard solution the algorithm $A_L(I)$ of problem L returns:

$$g(I,r) = \begin{cases} u \text{ if } r \in \mathscr{R} \\ A_L(I) \text{ otherwise} \end{cases}$$

Now by defining the cost function c_Q in a certain way, it allows us to transform the neighborhood structure from Hamming distance greater than 1 between the neighbors

to the Hamming distance of exactly 1, which is needed because the Min-circuit/Flip problem only deals with neighbors of Hamming distance 1. If u and w are neighbors in L, then the cost function leads to a sequence of neighbors from $uu00$ to $ww00$.

The general idea of the proof is to show that a local optimum in Q must have the form $uu00$ and that it follows that u is a local optimum in L.

If $uu00$ is not a local optimum in Q, it has to have a neighbor $ww00$ with better cost. We can show that if $uu00$ is not a local optimum, a sequence of better neighbors will be chosen which leads to a solution $ww00$ with better cost, which eventually is a local optimum. This will follow by the definition of the cost function. Every other solution that has not the form $uu00$ has worse cost. If $ww00$ is a local optimum, w is a local optimum too. The last two bits in a solution $r \in F_Q(I)$ represent the state on the way of transforming $uu00$ to $ww00$ and are needed to define the cost function correctly.
The cost function is defined as follows:

Costs for "well structured" solutions:

$$
\begin{aligned}
c_Q(f(I), uu00) &= (2p+4)c_L(I,u) \\
c_Q(f(I), uv00) &= (2p+4)c_L(I,w) + (p+2) + H(v,w) + 2 \\
&\quad \text{where } u \neq v \text{ and } w \text{ is the best neighbor of } u \text{ in } L \text{ returned by } C_L \\
&\quad H(v,w) \text{ is the Hamming distance between } v \text{ and } w \\
&\quad \text{Note: } c_Q(I, uw00) \text{ is included in this definition of the cost, when } v = w \\
&\quad \text{then } H(v,w) = 0 \text{ and } c_Q(I, uw00) = (2p+4)c_L(I,w) + (p+2) + 2 \\
c_Q(f(I), uw10) &= (2p+4)c_L(I,w) + (p+2) + 1 \\
c_Q(f(I), uw11) &= (2p+4)c_L(I,w) + H(u,w) + 2 \\
c_Q(f(I), vu11) &= (2p+4)c_L(I,u) + H(v,u) + 2 \\
c_Q(f(I), uu11) &= (2p+4)c_L(I,u) + 2 \\
c_Q(f(I), uu10) &= (2p+4)c_L(I,u) + 1
\end{aligned}
$$

Cost for any other solution, that is not in $\mathscr{R}$ or on the path between one element in $\mathscr{R}$ to another:

$$
\begin{aligned}
c_Q(f(I), s) &= Z + H(s, aa00) \\
&\quad \text{with } Z = (4p+4)2^{q(|I|)} \text{ for a polynomial function } q \\
&\quad \text{and the solution } a \text{ of } L \text{ returned by } A_L. \text{ [MAK07], [Yan88]}
\end{aligned}
$$

Therefore, every other solution that is not a well structured solution will be sequentially changed to $aa00$, as the cost is only Z if $s = aa00$, and more otherwise. From there on, a local optimum can be found.

So if we now have a solution of the form $uu00$ which is not a local optimum, the cost function will lead to a better neighbor.

Table 5: Sequence of chosen neighbors starting with $uu00$

Solution	Cost
$uu00$	$(2p+4)c_L(I,u)$
$\downarrow$	
uv_100	$(2p+4)c_L(I,w)+(p+2)+H(v_1,w)+2$
$\downarrow$	
...	
$\downarrow$	
$uv_k00 = uw00$	$(2p+4)c_L(I,w)+(p+2)+2$
$\downarrow$	
$uw10$	$(2p+4)c_L(I,w)+(p+2)+1$
$\downarrow$	
$uw11$	$(2p+4)c_L(I,w)+H(u,w)+2$
$\downarrow$	
v'_1w11	$(2p+4)c_L(I,w)+H(v'_1,w)+2$
$\downarrow$	
...	
$\downarrow$	
$v'_kw11 = ww11$	$(2p+4)c_L(I,w)+2$
$\downarrow$	
$ww10$	$(2p+4)c_L(I,w)+1$
$\downarrow$	
$ww00$	$(2p+4)c_L(I,w)$

Changing the second u to v_1 so that $H(u,v_1)=1$ is the best possible option as this costs $(2p+4)c_L(I,w)+(p+2)+H(v_1,w)+2$ instead of $(2p+4)c_L(I,u)$. We do not change the first u, as $v_1 \notin F_L(I)$ because $H(u,v_1)=1$ but all solutions in L have Hamming distance of at least 2.
w is by definition the next best neighbor of u in L. This leads to uv_100 with $H(uu00, uv_100) = 1$.
We repeat this procedure k times, until we reach $uv_k00 = uw00$. $H(v_k,w)$ is now 0, as $v_k = w$.
The cost function implies that it is now the best to choose $uw10$ with cost $(2p+4)c_L(I,w)+(p+2)+1$ which is one less than the cost of $uw00$.
$uw11$ is chosen next, $H(v,w)+2$ is smaller than $(p+2)+1$ as p is the length of u and w, and their Hamming distance cannot be greater than p. So the cost $c_Q(I,uw11) = (2p+4)c_L(I,w)+H(v,w)+2$ is smaller than $c_Q(I,uw10) = (2p+4)c_L(I,w)+(p+2)+1$.
Now the first u is transformed step by step to w too, again by using $v'_1...v'_k$. We then have $ww11$ with cost $c_Q(I,ww11) = (2p+4)c_L(I,w)+2$.
The next neighbor is $ww10$ with cost $c_Q(I,ww10) = (2p+4)c_L(I,w)+1$ which is one less than $c_Q(I,ww11)$.
The same holds for the next neighbor $ww00$ with cost $c_Q(I,ww00) = (2p+4)c_L(I,w)$.

We now reached the encoded best neighbor of u, starting by the encoding of u. If $ww00$ is not a local optimum, we will go through this sequence again and result in a solution $xx00$. If $xx00$ really is better than $ww00$, then x is better than w too. So if we finally reach a local optimum $ww00$ of Q, then w is a local optimum of L.

The cost function can be calculated in polynomial time.

We defined f, g, a neighborhood structure for Q and a cost function for Q out of any PLS-complete problem, all in polynomial time.

Lemma 3.3 *This reduction is tight.*

Proof. This is proven in [Yan88, p.44].

Lemma 3.4 $Q \preceq$ *Min-circuit/Flip.*

Proof. Since Q is in PLS, it has a cost function c_Q that can be calculated in polynomial time. This cost function is now used to define the function f of the PLS-reduction. There exists a Turing machine that builds a boolean circuit out of the cost function in polynomial time [MAK07, p.104]. The input of the boolean circuit is any solution x_1, ..., x_{2p+2} of Q with length $2p + 2$. The outputs y_1, ..., y_m represent the cost of x_1, ..., x_{2p+2} in binary.

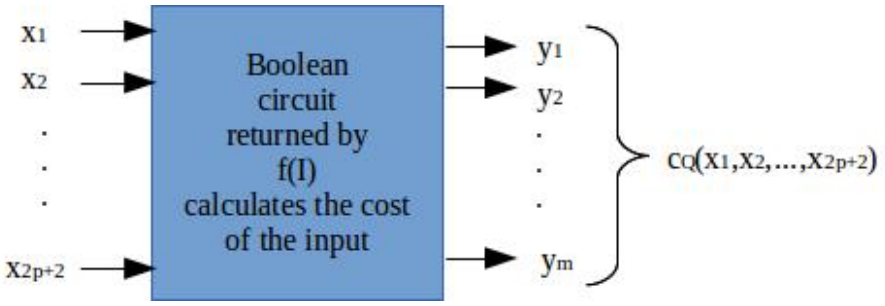


Figure 5: Construction of $I_{circuit/Flip} = f(I_Q)$

The function g of the PLS-reduction is the identity.

The local and global optima of Q and Min-circuit/Flip are obviously the same: If a solution s of Q is locally optimal for Q, none of its neighbors has better costs. As the costs of Min-circuit/Flip are exactly the same as for Q, as well as their neighborhood structure, an optimum of Q is an optimum of Min-circuit/Flip too.

Lemma 3.5 *This reduction is tight.*

Proof. This is proven in [Yan88, p.44].

4. PLS-complete Problems

In this chapter an overview of PLS-complete problems is presented. Every problem is defined, and a proof of its PLS-completeness is referenced. A few proofs are presented here too: Positive-not-all-equal-max-3Sat/Kernighan-Lin, Max-Uniform-Graph-partitioning/Kernighan-Lin, Max-cut/Flip and Max-2Sat/Flip. They show the basic idea of how a reduction proceeds.

Positive-not-all-equal-max-3Sat/Kernighan-Lin is proven PLS-complete by a quite difficult reduction from Max-circuit/Flip. The original paper [JPY88] includes this proof when they reduce from Max-circuit/Flip to Max-Uniform-Graph-partitioning/Kernighan-Lin. Here, as well as in [Yan88], the proof has been split up, as there are other problems that can be reduced from Positive-not-all-equal-max-3Sat/Kernighan-Lin, like Max-cut/Kernighan-Lin.

Max-Uniform-Graph-partitioning/Kernighan-Lin is proven PLS-complete by reduction from Positive-not-all-equal-max-3Sat/Kernighan-Lin and shows us how we can reduce a problem with clauses to a graph problem.

Max-cut/Flip and Max-2Sat/Flip are two examples of simpler reductions.

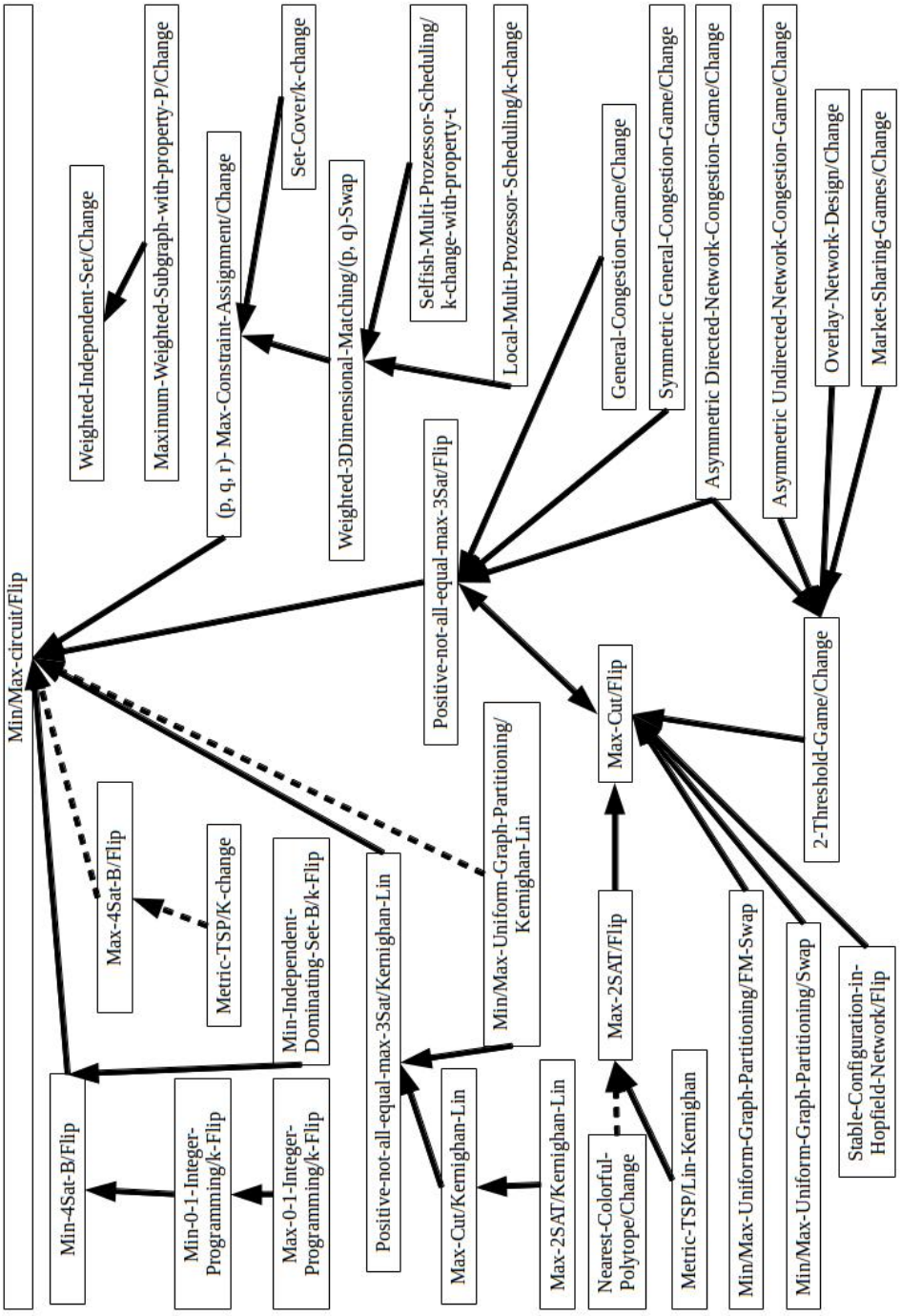


Figure 6: Overview of PLS-complete problems and how they are reduced to each other
Syntax: Optimization-Problem/Neighborhood structure
Dotted arrow: PLS-reduction from a problem L to a problem Q: $L \leftarrow Q$
Black arrow: Tight PLS-reduction

Table 6 on the following page: Alphabetic overview of the following PLS-complete problems:

Name	Neighborhood	Reference	Section
2-Threshold-Games	Change	[ARV08]	4.15 on page 51
Asymmetric Network-Congestion-Games	Change	[ARV08], [FPT04]	4.14 on page 49
Asymmetric Undirected-Network-Congestion-Games	Change	[FPT04]	4.14 on page 49
General-Congestion-Games	Change	[FPT04]	4.13 on page 48
Local-Multi-Processor-Scheduling	k-Change	[JPY88]	4.11 on page 47
Market-Sharing-Games	Change	[ARV08]	4.16 on page 51
Max-2Sat	Flip Kernighan-Lin	[SY91] Claimed in [MAK07]	4.2 on page 35
Max-Cut	Flip Kernighan-Lin	[SY91] Claimed in [MAK07]	4.5 on page 41
Maximum-Weighted-Subgraph-with-property-P	Change	[Shi97]	4.8 on page 44
Metric-Traveling-Salesman-Problem	k-change Lin-Kernighan	[Kre89] [PSY90]	4.10 on page 45
Min-Independent-Dominating-Set-B	k-Flip	[Kla96]	4.6 on page 43
Min/Max-0-1-Integer-Programming	k-Flip	[Kla96]	4.20 on page 53
Min/Max-4Sat-B	Flip	[Kla96]	4.3 on page 36
Min/Max-circuit	Flip	[JPY88]	3 on page 15
Min/Max-Uniform-Graph-Partitioning	Swap Kernighan-Lin Fiduccia-Matheyses FM-Swap	[SY91] [JPY88], [Yan88] stated without proof in [Yan88] [SY91]	4.4 on page 37
Nearest-Colorful-Polytope	Swap	[MS14]	4.19 on page 53
Overlay-Network-Design	Change	[ARV08]	4.17 on page 52
Positive-not-all-equal-max-3Sat	Flip Kernighan-Lin	[SY91] [Yan88]	4.1 on page 25

(p, q, r)-Max-Constraint-Assignment-k-parite	Change	[DM13]	4.21 on page 54
Selfish-Multi-Processor-Scheduling	k-change-with-property-t	[DMT09]	4.12 on page 48
Set-Cover	k-Change	[DS10]	4.9 on page 45
Stable-Configuration in a Hopfield network with threshold=0 and negative weights	Flip	[SY91], [PSY90], [Yan88]	4.18 on page 52
Symmetric-General-Congestion-Games	Change	[FPT04]	4.13 on page 48
Weighted-3Dimensional-Matching	(p, q)-Swap	[DMT09]	4.22 on page 55
Weighted-Intependent-Set	Change	Stated without proof in [SY91]	4.7 on page 43

Table 6: Alphabetic overview of the following PLS-complete problems

4.1. Positive-not-all-equal-max-3Sat

Definition 4.1 An instance I consists of a set of binary variables $U = \{x_1, ..., x_n\}$ and a set of clauses C, where one clause $c \in C$ is $c = NAE(y_1, ..., y_k)$. An y_i can be a constant 0 or 1, a variable $x_j \in U$ or its negation $\neg x_j$. The number of literals in one clause is at most three, so $1 \leq k \leq 3$. The weight $w(c)$ of a clause $c \in C$ is a positive integer.

A clause c is satisfied, if at least one literal is true and one literal is false. The aim is to find an assignment that maximizes the sum of the weights of the clauses that are satisfied. So the cost of a solution s, which is an assignment of the variables as a bit string, is the sum of the weights of the satisfied clauses. Lets consider the problem under the two neighborhood structures Flip and Kernighan-Lin.

Flip The neighbor r of a solution s can be achieved by negating (flipping) one input bit x_i. So one solution s and all its neighbors $r \in N(I, s)$ have Hamming distance one: $H(s, r) = 1$.

Kernighhan-Lin A solution r is a neighbor of solution s if r can be obtained from s by a sequence of greedy flips, where no bit is flipped twice. This means, starting with s, we choose the flip neighbor s_1 of s, with the best cost, or the least loss of cost, to be a neighbor of s in the Kernighan-Lin structure. As well as best (or least worst) neighbor of s_1, and so on, until s_i is a solution where every bit of s is negated. Note that we are not allowed to flip a bit back, if it once has been flipped.

Example 4.1 Consider the following instance of NAE $c_1 = \{x_1, x_2\}$, $c_2 = \{x_2, \neg x_3\}$ and $c_3 = \{x_1, 0\}$ with weight $w(c_1) = w(c_2) = w(c_3) := 1$. Let the initial solution be $(0, 0, 0)$ with cost 1 which means that for $x_1 = 0, x_2 = 0$ and $x_3 = 0$ only one clause with weight 1 is satisfied, which is the second one. The neighbors of $(0, 0, 0)$ under the Kernighan-Lin neighborhood structure are all elements on the path of greedy flips, until no bit can be flipped any more, as they are only allowed to be flipped once.

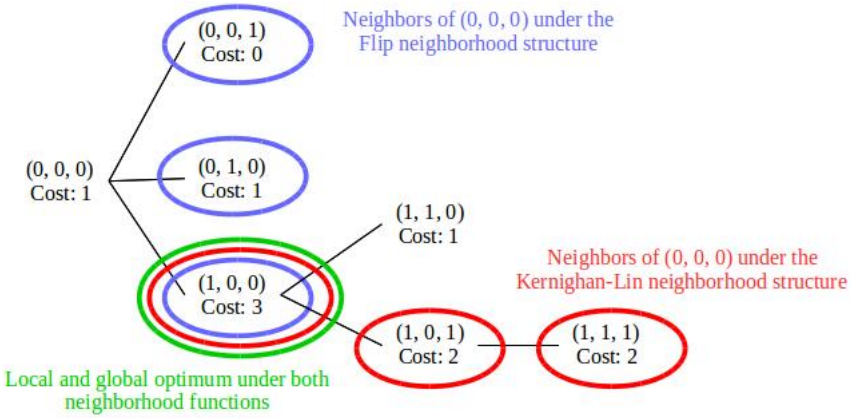


Figure 7: Kernighan-Lin neighborhood compared to Flip neighborhood

So the Kernighan-Lin neighbors of $(0,0,0)$ are $(1,0,0)$, $(1,0,1)$ and $(1,1,1)$. $(1,0,0)$ is the best Flip neighbor of $(0,0,0)$, as it has a weight of 3. $(1,0,1)$ is the best Flip neighbor of $(1,0,0)$, as it is the neighbor with the least bad cost. $(1,1,1)$ is the best Flip neighbor of $(1,0,1)$. There are no more neighbors in the sequence after $(1,1,1)$, because each bit has been flipped already once. In the next iteration, to find the neighbors of $(1,0,0)$ each bit is allowed to be flipped once again.

If the Kernighan-Lin structure stopped at the sequence after it found a local optimum under the Flip neighborhood structure, it would be like the Flip neighborhood, except that it would skip the solutions that are not a local optimum. It can happen though, that there is a solution s that is a local optimum under the Flip neighborhood, but not under the Kernighan-Lin neighborhood. Imagine s is at the beginning of the sequence of Flips, Flip would stop at s and return it as a local optimum. Kernighan-Lin keeps flipping, even though the costs might be worse than the cost of its predecessor in the sequence, but maybe at the end of the sequence, there is another local optimum r of Flip, with r having better costs than s. Therefore Kernighan-Lin returns r. The local optimum of the Kernighan-Lin structure is always a local optimum of the Flip structure too, but a local optimum of the Flip structure is not necessarily a local optimum of the Kernighan-Lin structure.

Theorem 4.1 *Positive-not-all-equal-max-3Sat/Flip is PLS-complete.*

Proof. [SY91, p.75] proved PLS-completeness via a tight PLS-reduction from Min/Max-circuit/Flip to Positive-not-all-equal-max-3Sat/Flip. Positive-not-all-equal-max-3Sat/Flip is in PLS.

Note that [SY91] proved that Positive-not-all-equal-max-3Sat/Flip can be reduced from Max-Cut/Flip too.

Theorem 4.2 *Positive-not-all-equal-max-3Sat/Kernighan-Lin is PLS-complete.*

Proof. Positive-not-all-equal-max-3Sat/Keringhan-Lin is in PLS as there exist all three algorithms A, B and C that run in polynomial time:

- $A(I)$ produces any solution. As the length of any solution is polynomial bounded in the number of input bits, A runs in polynomial time.
- $B(I, s)$ calculates the cost of a solution, by summing up the weights of the satisfied clauses, which can be done in polynomial time too, as the number of clauses is polynomial bounded in $|I|$.
- $C(I, s)$ searches the neighborhood for a better neighbor. The set $N(I, s)$ has size n, as it contains all solution with one more of the n input bits flipped. Therefore, C can be computed in polynomial time too.

Furthermore Min/Max-circuit/Flip, which is a PLS-complete problem, can be tightly PLS-reduced to Positive-not-all-equal-max-3Sat/Kernighan-Lin. [Yan88]

For the reduction we will use Max-circuit/Flip. An instance I of Max-circuit/Flip has $x_1, ..., x_n$ input bits, $y_1, ..., y_m$ output bits and a boolean circuit D'' consisting of *and*, *or* and *not* gates. For this reduction f will first transform the given boolean circuit D'' to a boolean circuit D' that only consists of *nor* gates, where:

$\neg a \wedge \neg b$	$nor(a, b)$
$\neg a$	$nor(a, 0)$
$a \wedge b$	$nor(\neg a, \neg b)$
$a \vee b$	$\neg nor(a, b)$

Every *nor* clause $nor(a, b)$ is true if and only if $a = 0$ and $b = 0$.

The size of D'' and D' only differs in a constant factor, and D'' can be transformed to D' in polynomial time [MAK07]. To use only *nor* gates makes it easier to construct the NAE-clauses, but the proof would work with *and*, *or* and *not* gates too (as done in [JPY88]).

Furthermore an additional layer of logic is added to extend the circuit D' to a circuit D, so that a vector $z \in \{0, 1\}^n$ is computed. It represents a better Flip neighbor of a solution s, if there is a better Flip neighbor, otherwise, if s is an local optimum, it is s itself. z can be computed by adding n copies of D', one for each flipped input variable x_i, and evaluating the circuit for every of the n possible neighbors of s. The input of the circuit D' with the best output is returned as $z_1, ..., z_n$. [MAK07]

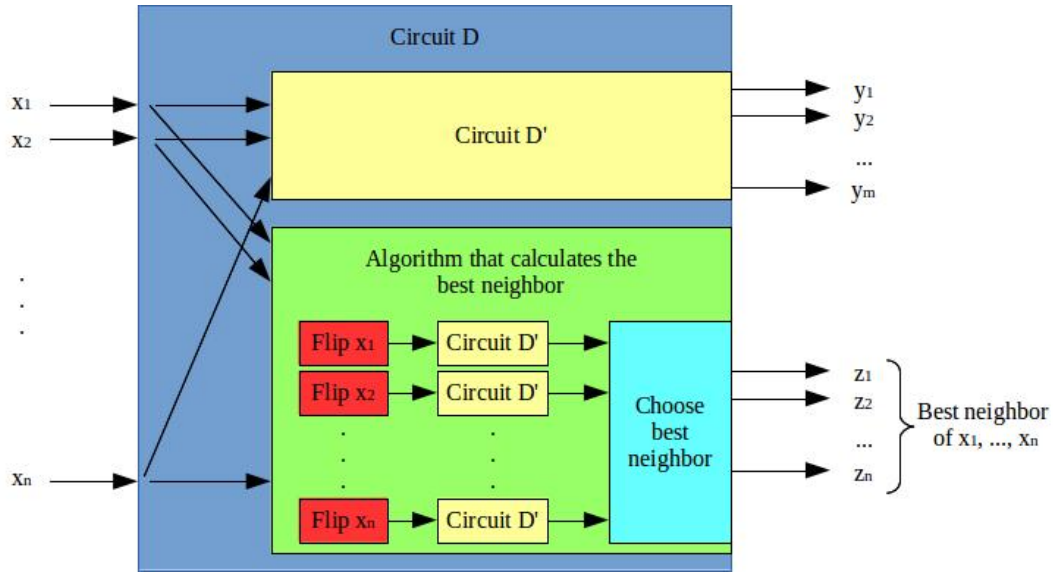
As will be explained later, we need the variables z_i to make the Kernighan-Lin neighborhood structure work properly and to ensure that the local optima of Positive-not-all-equal-max-3Sat/Kernighan-Lin are locally optimal in circuit/Flip too.

Example 4.2

$$D'' = \begin{pmatrix} x_1 \wedge x_2 \\ (x_1 \vee x_2) \wedge x_3 \end{pmatrix}$$

$$D' = \begin{pmatrix} nor(\neg x_1, \neg x_3) \\ nor(\neg\neg nor(x_1, x_2), \neg x_3 \end{pmatrix} = \begin{pmatrix} nor(nor(x_1, 0), nor(x_3, 0)) \\ nor(nor(x_1, x_2), nor(x_3, 0)) \end{pmatrix}$$

Figure 8: Circuit D



So an instance I of Max-circuit/Flip has $x_1, ..., x_n$ input bits, $y_1, ..., y_m$ output bits, $z_1, ..., z_n$ additional output bits representing a better Flip neighbor and boolean circuit D consisting of $|D|$ *nor* gates.

Let $|D| \geq 4$.

The function f of the reduction now constructs the following NAE-clauses. For each input variable x_i with $i = 1, ..., n$ there is a x'_{i1} and a x'_{i2}. They represent the negation of x_i, and we choose the weight of the following two clauses so that we gain the most weight if x'_{i1} and x'_{i2} are the negation of x_i:

NAE-clause	Weight
$\{x_i, x'_{i1}\}$	$M := (3\|D\|)^{(3\|D\|)}$
$\{x_i, x'_{i2}\}$	M

We introduce a gate variable g_j for each $nor(a, b)$ with $1 \leq j \leq |D| - n$, where g_j represents the output of the gate. In a "consistent" solution, if $nor(a, b)$ is the j'th

clause and the value of $nor(a, b)$ is 1, then g_j is 1. If $nor(a, b)$ is 0 then g_j is 0 [Yan88]. There is a variable g'_j, which represents the negation of g_j as well, and four clauses, where a and b are either a constant, another gate variable g_l or an input variable x_i. Therefore these clauses are called consistency clauses.

NAE-clause	Weight
$\{g_j, g'_j\}$	M
$\{g_j, a, b\}$	$N_j := \frac{M}{(3\|D\|)^{2j}}$
$\{g_j, a, 1\}$	N_j
$\{g_j, b, 1\}$	N_j

For each output node y_k where $k = 1, ..., m$ we define the following clause. It represents the output variables of the circuit/Flip instance I. The weight of this clause is the same as the variable y_k gains in I.

NAE-clause	Weight
$\{y_k, 0\}$	$2^{(k-1)}$

For each z_i, where $i = 1, ..., n$ there is a clause comparing the i'th bit of the better neighbor solution to its original bit x_i. This represents whether solution $x_1, ..., x_n$ has a better neighbor for the circuit/Flip instance I under the Flip neighborhood function.

NAE-clause	Weight
$\{x'_{i1}, z_i\}$	2

Note that $y_1, ..., y_m$ and $z_1, ..., z_n$ are gate variables $g_{z_i} = z_i$ and $g_{y_k} = y_k$, as they are the last gates in the circuit. There are clauses with z_i and clauses with y_k, but for every $z_i = g_{z_i}$ there are the consistency clauses and a clause $\{g_{z_i}, g'_{z_i}\}$ as well. So flipping a z_i means changing a consistency clause too. The same holds for each y_k.

Furthermore we add one clause with a new variable p.

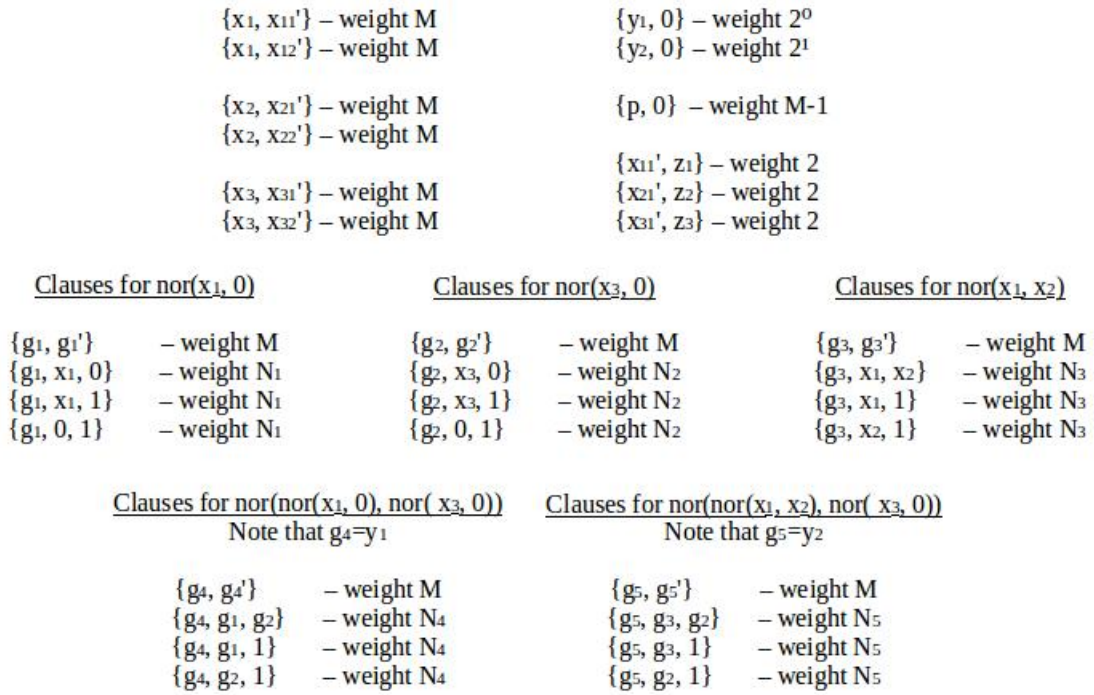
NAE-clause	Weight
$\{p, 0\}$	$M - 1$

Example 4.3 of the constructed NAE-clauses of instance $I_{NAE} = f(I_{circuit/Flip})$

Encoded circuit $I_{circuit/Flip}$ of circuit/Flip:

$$\begin{pmatrix} nor(nor(x_1, 0), nor(x_3, 0)) \\ nor(nor(x_1, x_2), nor(x_3, 0)) \end{pmatrix}$$

Figure 9: D' of $I_{NAE} = f(I_{circuit/Flip})$:



The function g of the reduction is defined as follows:
$g(I_{circuit/Flip}, (x_1, ...,x_n, x'_{11}, x'_{12}, ...,x'_{n1}, x'_{n2}, g_1, ..., g_{|D|-n}, g'_1, ..., g'_{|D|-n}, p)) = (x_1, ...,x_n)$
$y_1, ...,y_m$ and $z_1, ..., z_n$ are included in the gate variables g_i.

We call a well structured solution s a solution where:

- $x'_{i1} = x'_{i2} \neq x_i$
- $g_i \neq g'_i$
- $p = 1$
- The variables x_i and g_i define a valid computation of the boolean circuit D, which leads to all consistency clauses being satisfied

[MAK07, p.111]

In a locally optimal solution, the consistency clauses are always satisfied:
If the NAE clauses do not correspond with a valid assignment of the *nor* clause, there is at least one of the three clauses that is not satisfied, but that could be satisfied by flipping one of the three variables, and so the variable will be flipped in a better neighboring solution. Any clause that contains g_j, a or b as an input and becomes unsatisfied by flipping one of the variables, can be adjusted like that as well. As there are no cycles,

the adjustment is propagated upwards through the clauses and stops if it reaches the clause with the highest weight. [Yan88, p.38, 39 (ii)]

There is always a path of better neighbors from a non well structured solution to a better well structured solution. It is always best to set the variable p to one, as we gain weight $M-1$ and there is no other clause that could lose weight, because no other clause contains p. x_i will have a different value than x'_{i1} and x'_{i2}, as we gain weight of $2M$ if they do not equal x_i. As x'_{i1} and x'_{i2} are in no other clause (except $\{x'_{i1}, z_i\}$ which only has weight 2), no other clause is affected when we flip the value of x'_{i1} and x'_{i2}, therefore we can do that without any loss of weight. We can argue that after flipping a gate variable g_j, the next greedy flip is to flip g'_j. g'_j, too, never appears in any other clause, so there is no harm done in flipping g'_j. [MAK07, p.116, 117, Proof of condition 2]

The cost of a well structured solution is always higher than the cost of a non well structured solution. Therefore Lemma 4.1 follows.

Lemma 4.1 *A local optimum is always a well structured solution.*

The path from one well structured solution to a better well structured solution:
If we have a well structured solution s, we look at all the neighbors of s to see whether or not there is one with better costs. A neighbor r of s can be reached by a sequence of greedy flips. The general idea is to show that this sequence always flips the elements in the same order.

Step 1: Flip x'_{q1}
The first of these greedy flips is to flip a x'_{q1} where $x'_{q1} = z_q$, which loses weight M but gains weight 2. Flipping any of the other variables would lead to a worse result:

Flipping this variable	leads to losing this clause/s	with weight
x'_{q1} where $z_q = x'_{q1}$	$\{x_q, x'_{q1}\}$ but gains $\{x'_{q1}, z_q\}$	$M-2$
x'_{i1} where $z_i \neq x'_{i1}$	$\{x_i, x'_{i1}\}, \{x'_{i1}, z_i\}$	$M+2$
$g_j, z_i \neq g_j, y_k \neq g_j$	$\{g_j, g'_j\}$ and at least one of the consistency clauses	$\geq M + N_j$
$g_j, z_i = g_j$ and/or $y_k = g_j$	$\{g_j, g'_j\}$ and at least one of the consistency clauses and we gain one or more clauses of the form $\{y_k, 0\}$ and $\{x'_{i1}, z_i\}$ (maximal all of them)	$\geq M + N_j - 3^{\lvert D\rvert}$ $> M-2$ as $N_j > 3^{\lvert D\rvert}$ [A3],[A4]
g'_j	$\{g_j, g'_j\}$	M
p	$\{p, 0\}$	$M-1$
x'_{i2}	$\{x_i, x'_{i2}\}$	M
x_i	$\{x_i, x'_{i1}\}, \{x_i, x'_{i2}\}$, and maybe consistency clauses	$\geq M + M$

Table 7: Flip x'_{q1}

Step 2: Flip x_q
After flipping x'_{q1}, the next greedy flip is to flip the corresponding x_q, because then we gain the weight of clause $\{x_q, x'_{q1}\}$ again, even though we lose the weight of $\{x_q, x'_{q2}\}$ and possibly some consistency clauses. But all other options would be worse again:

Flipping this variable	leads to losing this clause/s	with weight
x_q	$\{x_q, x'_{q2}\}$ but gain $\{x_q, x'_{q1}\}$ and maybe lose some consistency clauses: weight of any consistency clause $\leq N_1 = (3\|D\|)^{(3\|D\|-2)}$, at most three clauses per gate, at most $\|D\| - n$ gates	$\leq M - M +$ $(\|D\| - n) \cdot 3 \cdot N_1$ $< M - 2$ [A5]
$x_i, i \neq q$	$\{x_i, x'_{i1}\}, \{x_i, x'_{i2}\}$, and maybe consistency clauses	$\geq M + M$
$x'_{i1}, i \neq q, x'_{i1} \neq z_i$	$\{x_i, x'_{i1}\}, \{x'_{i1}, z_i\}$	$M + 2$
$x'_{i1}, i \neq q, x'_{i1} = z_i$	$\{x_i, x'_{i1}\}$, but we gain $\{x'_{i1}, z_i\}$	$M - 2$
x'_{i2}	$\{x_i, x'_{i2}\}$	M
$g_j, z_i \neq g_j, y_k \neq g_j$	$\{g_j, g'_j\}$ and at least one of the consistency clauses	$\geq M + N_j$
$g_j, z_i = g_j$ and/or $y_k = g_j$	$\{g_j, g'_j\}$ and at least one of the consistency clauses and we gain one or more clauses of the form $\{y_k, 0\}$ and $\{x'_{i1}, z_i\}$ (maximal all of them)	$\geq M + N_j - 3^{\|D\|}$ $> M - 2$ as $N_j > 3^{\|D\|}$ [A3],[A4]
g'_j	$\{g_j, g'_j\}$	M
p	$\{p, 0\}$	$M - 1$

Table 8: Flip x_q

Step 3: Flip x'_{q2}
The next greedy flip flips x'_{q2}, as we gain the weight M and lose nothing. Since M is a greater gain than any N_j that we might gain by flipping g_j, we flip x'_{q2}. Flipping anything else would lead to a loss of weight. Note that x_q and x'_{q1} are not allowed to be flipped again, as they have been already flipped once. By this we gain more than we would gain by any other flip:

Flipping this variable	leads to losing this clause/s	with weight
x'_{q2}	Nothing, but we gain $\{x_q, x'_{q2}\}$	$-M$
$x'_{i2}, i \neq q$	$\{x_i, x'_{i2}\}$	M
$x'_{i1}, i \neq q, x'_{i1} \neq z_i$	$\{x_i, x'_{i1}\}, \{x'_{i1}, z_i\}$	$M + 2$
$x'_{i1}, i \neq q, x'_{i1} = z_i$	$\{x_i, x'_{i1}\}$, but we gain $\{x'_{i1}, z_i\}$	$M - 2$
$x_i, i \neq q$	$\{x_i, x'_{i1}\}, \{x_i, x'_{i2}\}$, and maybe a consistency clause	$\geq M + M$

$g_j, z_i \neq g_j, y_k \neq g_j$ clause j doesn't contain x_q	$\{g_j, g'_j\}$ and at least one of the consistency clauses	$\geq M + N_j$
$g_j, z_i = g_j$ and/or $y_k = g_j$ clause j doesn't contain x_q	$\{g_j, g'_j\}$ and at least one of the consistency clauses and we gain one or more clauses of the form $\{y_k, 0\}$ and $\{x'_{i1}, z_i\}$ (maximal all of them)	$\geq M + Nj - 3^{\lvert D\rvert}$ $> M$ as $N_j > 3^{\lvert D\rvert}$ [A3], [A4]
$g_j, z_i \neq g_j, y_k \neq g_j$ clause j contains x_q	$\{g_j, g'_j\}$, we gain at most three consistency clauses	$\geq M - 3 \cdot N_j$ $> 0 > -M$ as $M > 3 \cdot N_j$ [A1]
$g_j, z_i = g_j$ and/or $y_k = g_j$ clause j contains x_q	$\{g_j, g'_j\}$ and we gain at most three consistency clauses and one or more clauses of the form $\{y_k, 0\}$ and $\{x'_{i1}, z_i\}$ (maximal all of them)	$\geq M - 3 \cdot N_j - 3^{\lvert D\rvert}$ $> -M$ [A2]
g'_j	$\{g_j, g'_j\}$	M
p	$\{p, 0\}$	$M - 1$

Table 9: Flip x'_{q2}

Step 4:
Now, for every inconsistent consistency clause first g_j and then g'_j is flipped. Flipping g_j leads to a loss of at most $M - 3N_j - 3^{|D|}$. Furthermore we lose at most $3(|D| - n - j)N_{j+1}$ constistency clauses of all consitency clauses greater than j, as we start flipping the g_j with the smallest j first. It will start with the smallest clause, where N_j is the highest if j is the smallest.
Flipping anything else than g_j leads to a higher loss:

Flipping this variable	leads to losing this clause/s	with weight
g_j, clause j is inconsistent and contains x_q	$\{g_j, g'_j\}$, we gain at most all three consistency clauses, and lose at least $3(\lvert D\rvert - n - j)$ consistency clauses that contain g_j and we gain one or more clauses of the form $\{y_k, 0\}$ and $\{x'_{i1}, z_i\}$ (maximal all of them)	$\leq M - 3N_j + 3(\lvert D\rvert - n - j)N_{j+1} - 3^{\lvert D\rvert}$ $< M - 2$ [A3],[A6]
g_j, clause j is consistent, $z_i \neq g_j, y_k \neq g_j$	$\{g_j, g'_j\}$ and at least one of the consistency clauses	$\geq M + N_j$
g_j, clause j is consistent, $z_i = g_j$ and/or $y_k = g_j$	$\{g_j, g'_j\}$ and at least one of the consistency clauses and we gain one or more clauses of the form $\{y_k, 0\}$ and $\{x'_{i1}, z_i\}$ (maximal all of them)	$\geq M + N_j - 3^{\lvert D\rvert}$ $> M$ as $N_j > 3^{\lvert D\rvert}$ [A3],[A4]
g'_j	$\{g_j, g'_j\}$	M
$x'_{i2}, i \neq q$	$\{x_i, x'_{i2}\}$	M
$x'_{i1}, i \neq q$	$\{x_i, x'_{i1}\}, \{x'_{i1}, z_i\}$	$M + 2$

$x'_{i1}, i \neq q, x'_{i1} \neq z_i$	$\{x_i, x'_{i1}\}$, but we gain $\{x'_{i1}, z_i\}$	$M-2$
$x_i, i \neq q$	$\{x_i, x'_{i1}\}, \{x_i, x'_{i2}\}$, and maybe a consistency clause	$\geq M+M$
p	$\{p, 0\}$	$M-1$

Table 10: Flip g_j

x'_{q2}, x_q and x'_{q1} are not allowed to be flipped again, as they have been flipped once already.

The next flip is g'_j, we gain weight M and lose nothing, this is better than flipping any other g_j, which at least loses some weight.

There is a g_j flipped as long as there are inconsistent clauses. If there is no inconsistent gate anymore, p is flipped to 0, and all following solutions are not well structured any more. The solution r where there is no inconsistent gate and $p = 1$ is the next best neighbor of the Kernighan-Lin neighborhood structure. The result r is a solution with better costs than s and better costs than any step on the sequence from s to r.
From there on, the algorithm can be performed again to reach an even better neighbor if one exists. Otherwise r is a local optimum.

flip x'_{i1};
flip x_i;
flip x'_{i2};
for $j = 1, j \leq |D| - n, j = j + 1$ **do**
  if g_j *is inconsistent* **then**
    flip g_j;
    flip g'_j;
  end
end

Algorithm 2: Algorithm for deriving the sequence of flips the Kernighan-Lin neighborhood performs to find a better neighbor of s [MAK07, p.114 Figure 6.6]

This results in a higher weight than before. We only start flipping if there is a $z_i = x'_{i1} \neq x_i$. This means that circuit/Flip has a better neighbor z_1, ..., z_n so that the output y_1, ..., y_m has a higher value. It follows that with z_1, ..., z_n as input more clauses in NAE would be satisfied: the clause $\{z_i, x'_{i1}\}$ and the clauses that represent y_k, so all consistency clauses with $g_{y_k} = y_k$ and $\{y_k, 0\}$.
We reach a local optimum if for all i $z_i \neq x'_{i1} \neq x_i$, which means that $x_i = z_i$ for every i. Flipping any variable except $p = 0$ will lead to a loss.

Lemma 4.2 *The local and global optima of Positive-not-all-equal-max-3Sat/Kernighan-Lin are local and global optima in the corresponding Max-circuit/Flip instance too.*

Proof. Assume solution s of Positive-not-all-equal-max-3Sat/Kernighan-Lin is a local optimum but $g(s)$ is not. Then there is a *nor* clause $nor(a, b)$ that is not satisfied, but which can be satisfied by flipping a or b. This flip will lead to a solution with higher costs, giving us a y_k that turns to 1. In other words, there is a neighbor r with better costs than $g(s)$. This means, that there is a z_i in $f(I)$ that differs from x_i, which leads to a sequence of flips that increases the cost of solution s and leads to a better solution q. Therefore s was not locally optimal. [Yan88, p.39]

So in a local optimal solution $x_i = z_i$ for every $i = 1, ..., n$. The most greedy flip the Kernighan-Lin neighborhood structure could do is flipping p with loss $M - 1$. This makes any neighboring solution not well structured. [MAK07, p.115]

The functions f and g can be computed in polynomial time.

Thus, Positive-not-all-equal-max-3Sat/Kernighan-Lin is PLS-complete.

4.2. Max-2Sat

Definition 4.2 Let U be a set of binary variables $x_1, ..., x_n$ and let C be a set of clauses c over U. The number of literals in each clause is at most two and the weight $w(c)$ of a clause $c \in C$ is a positive integer. A solution is an assignment of $x_1, ..., x_n$. A clause is satisfied, if at least one variable is 1.
The aim is to find an assignment, that maximizes the sum of the weights of the satisfied clauses [MAK07].

Flip A neighbor r of a solution s is obtained by flipping one bit x_i of s [MAK07], [Yan88].

Kernighan-Lin A neighbor r of a solution s is obtained by performing a sequence of greedy flips [MAK07].

Theorem 4.3 *Max-2Sat/Flip is PLS-complete.*

Proof. Max-2Sat/Flip is in PLS.

There is a valid PLS-reduction (f, g) from Max-Cut/Flip (See Section 4.5 on page 41), which is PLS-complete, to Max-2Sat/Flip:

Graph $G = (V, E)$ with $V = \{x_1, x_2, ..., x_n\}$ is an instance I of Max-Cut/Flip with weight $w(e) \in \mathbb{N}^+$ for all edges $e \in E$.

The function f constructs for every edge $e = (x_i, x_k) \in E$ two clauses $(x_i \vee x_k)$ and $(\neg x_i \vee \neg x_k)$, both of them with weight $w(e)$.

The function g gets a solution s which is an assignment of x_1, ..., x_n and returns a partition $(P0, P1)$, where all $x_i = 0$ are in one partition $P0$, and all $x_i = 1$ are in the other partition $P1$.

If $x_i = x_k$ there is only one of the two clauses with x_i and x_k satisfied. But both clauses are satisfied if $x_i \neq x_k$, so one of them is 0 and the other one is 1. Then we gain $2w(e)$ instead of just $w(e)$. This corresponds to one variable being in $P0$ and the other one in $P1$ which means we gain $w(e)$ instead of nothing. There is no other possibility, because either $x_i = x_k$ or $x_i \neq x_k$. Every time we gain $2w(e)$ in $f(I)$, we gain $w(e)$ in I. Every time we gain $w(e)$ in $f(I)$, we gain 0 in I. Hence a local optimum s in $f(I)$ is a local optimum $g(I, s)$ in I as well.

Therefore Max-2Sat/Flip is PLS-complete. [Yan88], [SY91]

This reduction is tight, as proven in [SY91, p.64].

Claim 4.1 *Max-2Sat/Keringhan-Lin is PLS-complete.*

This claim has been stated without proof in [MAK07]. [MAK07] also claims that PLS-completeness can be proven via a tight PLS-reduction from Max-Cut/Kernighan-Lin (See Section 4.5 on page 41) to Max-2Sat/Kernighan-Lin, where the reduction is identical to the one from Max-Cut/Flip to Max-2Sat/Flip. The construction is the same, and the local optima s in an instance $f(I)$ of Max-2Sat/Kernighan-Lin are local optima $g(I, s)$ in I of Max-Cut/Kernighan-Lin as well.

4.3. Min/Max-4Sat-B

Definition 4.3 Let an instance I be a boolean formula in conjunctive normal form with at most four literals in each clause. Furthermore, each variable x_1, ..., x_n is bounded in its occurrence by a constant B, so each x_i appears at most B times. Each clause has a certain weight $w(c)$, the cost of a solution is the sum of the weights of the satisfied clauses.

Flip A neighbor r of s is obtained by flipping an arbitrary input bit x_i.

Theorem 4.4 *Min-4Sat-B/Flip is PLS-complete.*

Proof. This has been proven in [Kla96, Lemma 2.1] via a tight PLS-reduction from Min-circuit/Flip to Min-4Sat-B/Flip. Min-4Sat-B/Flip is in PLS.

Theorem 4.5 *Max-4Sat-(B=3)/Flip is PLS-complete.*

Proof. [Kre89, p.218] proved PLS-completeness for Max-4Sat-(B=3)/Flip via a PLS-reduction from Max-circuit/Flip to Max-4Sat-(B=3)/Flip. Max-4Sat-(B=3) is in PLS. [Kre89] also claims without proof, that Max-C-Sat-B/Flip is PLS-complete. See also (p, q, r)-Max-Constraint-Assignment in Section 4.21 on page 54.

Theorem 4.6 *Max-4Sat-B/Flip is PLS-complete.*

Proof. [Kre90] proved PLS-completeness for Max-4Sat-B/Flip via a PLS-reduction from Max-circuit/Flip to Max-4Sat-B/Flip. Max-4Sat-B/Flip is in PLS.

Note that [Kre89] and [Kre90] call Max-4Sat-B/Flip "CNF-SAT".

4.4. Min/Max-Uniform-Graph-Partitioning

Definition 4.4 Let $G = (V, E)$ be a given graph with weights $w(e)$ on the edges $e \in E$. A solution s is any partition $(P0, P1)$ of V with $|P0| = |P1|$ and the cost $c(P0, P1)$ is the sum of the weights of the edges between $P0$ and $P1$, so the sum of the edges that have one endpoint in $P0$ and one endpoint in $P1$.

The optimal solution is a solution with minimum cost $c(P0, P1)$ if it is a minimization problem, and if it is a maximization problem the optimal solution is a solution with maximum cost $c(P0, P1)$.

Swap A partition $(P2, P3)$ is a neighbor of $(P0, P1)$ if $(P2, P3)$ can be obtained from $(P0, P1)$ by swapping one node $p0 \in P0$ with a node $p1 \in P1$.

Kernighan-Lin A partition $(P2, P3)$ is a neighbor of $(P0, P1)$ if $(P2, P3)$ can be obtained by a greedy sequence of swaps from nodes in $P0$ with nodes in $P1$ and vice versa. This means the two nodes $p0 \in P0$ and $p1 \in P1$ are swapped, where the partition $((P0 \setminus p0) \cup p1, (P1 \setminus p1) \cup p0)$ gains the highest possible weight, or loses the least possible weight. Note that no node is allowed to be swapped twice.

Fiduccia-Mattheyses This neighborhood is similar to the Kernighan-Lin neighborhood structure, it is a greedy sequence of swaps, except that each swap happens in two steps. First the $p0 \in P0$ with the most weight, or the least loss of weight, is swapped to $P1$, then the node $p1 \in P1$ with the most weight, or the least loss of weight is swapped to $P0$ to balance the partitions again.
Experiments have shown that Fiduccia-Mattheyses has a smaller runtime in each iteration of the standard algorithm, though it sometimes finds an inferior local optimum. [FM82], [Yan88, p.33]

FM-swap This neighborhood structure is based on the Fiduccia-Mattheyses neighborhood structure. Each solution $s = (P0, P1)$ has only one neighbor, the partition obtained after the first swap of the Fiduccia-Mattheyses. [Yan88, p.33]

Lemma 4.3 *Max-Uniform-Graph-partitioning and Min-Uniform-Graph-partitioning are equivalent.*

Proof. One instance I of one problem can easily be transformed to an instance I' of the opposite problem: Set the weights $w'(e)$ of the edges in I' to $w'(e) = W - w(e)$ where $W = \max_{e \in E} w(e)$ [Yan88, p.33]. Therefore, whenever we prove PLS-completeness for Max-Uniform-Graph-Partitioning under a certain neighborhood structure, the corresponding Min-Uniform-Graph-Partitioning problem is PLS-complete too.

Theorem 4.7 *Max-Uniform-Graph-Partitioning/Swap is PLS-complete.*

Proof. [SY91, Lemma 3.5 b)] proved Max-Uniform-Graph-Partitioning/Swap to be PLS-complete via a tight PLS-reduction from Max-Cut/Flip to Max-Uniform-Graph-partitioning/Swap. Max-Uniform-Graph-Partitioning/Swap is in PLS.

Claim 4.2 *Max-Uniform-Graph-Partitioning/Fiduccia-Matheyses is PLS-complete.*

This claim is stated like this without proof in [Yan88].

Theorem 4.8 *Max-Uniform-Graph-Partitioning/FM-Swap is PLS-complete.*

Proof. [SY91, Lemma 3.5 c)] proved Max-Uniform-Graph-Partitioning/FM-Swap to be PLS complete via a tight PLS-reduction from Max-Cut/Flip to Max-Uniform-Graph-Partitioning/FM-Swap. Max-Uniform-Graph-Partitioning/FM-Swap is in PLS.

Theorem 4.9 *Max-Uniform-Graph-Partitioning/Kernighan-Lin is PLS-complete.*

Proof. Max-Uniform-Graph-Partitioning/Kernighan-Lin is in PLS.

This Theorem was first proven by [JPY88] via a PLS-reduction from Min/Max-circuit/Flip to Max-Uniform-Graph-Partitioning/Kernighan-Lin. [Yan88] proved PLS-completeness via a tight PLS-reduction from Positive-not-all-equal-max-3Sat/Kernighan-Lin to Max-Uniform-Graph-Partitioning/Kernighan-Lin. We will now have a closer look at this proof.

There is a valid PLS-reduction (f, g) from the PLS-complete problem Positive-not-all-equal-max-3Sat/Kernighan-Lin to Max-Uniform-Graph-Partitioning/Kernighan-Lin:

The instance I^{NAE} of Positive-not-all-equal-max-3Sat/Kernighan-Lin consists of x_1^{NAE}, ..., x_n^{NAE} binary variables, and a set C of clauses. Each clause $c \in C$ has a weight $w(c)$ which is a positive integer.

The function f of the reduction constructs a graph $G = (V, E)$ of I^{NAE} in the following way:

Nodes:

- There is a node x_i and x'_i for every x_i^{NAE}, $1 \leq i \leq n$ of I^{NAE}, where x'_i stands for the negation of x_i.
- There are two nodes $T0$ and $T1$ that stand for the value 1.
- There are two nodes $F0$ and $F1$ that stand for the value 0.

Edges:

- There is an edge between every x_i and x'_i (the blue edges in the example below) with weight $W + 1$, where W is the total weight of all clauses in C.
- There are edges between $T0$ and $F0$, $T1$ and $F1$, $T0$ and $F1$, and $T1$ and $F0$ (the red edges in the example below), all of them with weight $W + 1$.
- For each clause $c =$NAE(x_p^{NAE}, x_q^{NAE}) there is an edge between the node x_p and x_q with weight $w(c)$.
- For each clause $c =$NAE$(x_p^{NAE}, x_q^{NAE}, x_r^{NAE})$ there are three edges, between x_p and x_q, x_q and x_r, as well as x_p and x_r. All of them have weight $\frac{w(c)}{2}$.
- If there is a constant 1 or a 0 in a clause, the corresponding node is $T0$ or $F0$.
- If there are several edges between two nodes x_p and x_q, there is only written one edge with the sum of the weights of all edges between them.

The function g transforms a solution $(P0, P1)$ of $f(I^{NAE})$ to an assignment of x_1^{NAE}, ..., x_n^{NAE} in the following way: $x_i^{NAE} = 1$ if $x_i \in Pk$ and $T0 \in Pk$, otherwise $x_i^{NAE} = 0$ [MAK07].

Example 4.4 of a locally and globally optimal solution:

- NAE$(x_1^{NAE}, x_2^{NAE}, 1)$ with $w($NAE$(x_1^{NAE}, x_2^{NAE}, 1)) = 6$ (the orange edges)
- NAE(x_2^{NAE}, x_3^{NAE}) with $w($NAE$(x_2^{NAE}, x_3^{NAE})) = 5$ (the green edge)
- Graph $G = (V, E)$ with $V = \{T0, T1, F0, F1, x_1, x_2, x_3, x'_1, x'_2, x'_3\}$
- $E = \{(T0, F0, W{+}1), (T1, F1, W{+}1), (T0, F1, W{+}1), (T1, F0, W{+}1), (x_1, x'_1, W{+}1), (x_2, x'_2, W + 1), (x_3, x'_3, W + 1), (x_1, x_2, 3), (x_1, T0, 3), (x_2, T0, 3), (x_2, x_3, 5)\}$
- $W + 1 = 12$

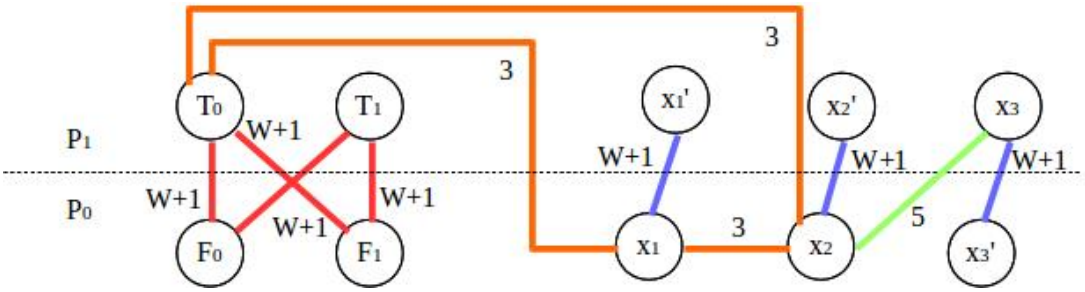


Figure 10: Optimal Partition

This partition ($P0 = \{F0, F1, x_1, x_2, x_3'\}, P1 = \{T0, T1, x_1', x_2', x_3\}$) is the global optimal solution for this problem instance. There is only one edge $(x_1, x_2, 3)$ that is not between the two partitions, and there is no way to include this edge without losing one or more of the other edges. The function g assigns the following values to the x_i^{NAE} : $x_1^{NAE} = 0, x_2^{NAE} = 0, x_3^{NAE} = 1$. This assignment also satisfies all Positive-not-all-equal-max-3Sat/Kernighan-Lin clauses.

All nodes x_i being in the partition with $T0$ have the value 1 in the corresponding assignment for I^{NAE}. Without loss of generality we call this partition $P1$. All nodes in the other partition $P0$ have the value 0 in the corresponding assignment. So a clause c in I^{NAE} is satisfied if the corresponding nodes are in different partitions, because this means, that one of them is 0 and one of them is 1, which is exactly what Positive-not-all-equal-max-3Sat/Kernighan-Lin asks for.

We introduced x_i' to every x_i, so we can make sure that the two partitions really can be equally sized, no matter if we have an even or an odd number of x_i^{NAE}'s. Without x_i', the assignment of all $x_i^{NAE} = 1$ could not be represented as a partition, because then all x_i would be in $P1$, and $P0$ would be smaller than $P1$. The nodes x_i' never have an edge other than to x_i. So it is always best to put them into the opposite partition than the corresponding x_i, which ensures that the two partitions have the same size and additionally gains the weight of $n \cdot (W + 1)$.

$T0$ and $T1$ will always be in the same partition, as well as $F0$ and $F1$, because that way we gain $4 \cdot (W + 1)$ weight where we would otherwise gain less. As $T1$ and $F1$ too have no edges other than to $T0$ and $F0$, we lose no weight by arranging them that way.

The solutions that are locally optimal in Max-Uniform-Graph-Partitioning/Kernighan-Lin are locally optimal in Positive-not-all-equal-max-3Sat/Kernighan-Lin too.
To prove this, let us assume that $(P0, P1)$ is a locally optimal partition, but $g(I^{NAE}, (P0, P1))$ is not a locally optimal solution for Positive-not-all-equal-max-3Sat/Kernighan-Lin.
It follows that there is a neighbor of $g(I^{NAE}, (P0, P1))$ with higher costs than $g(I^{NAE}, (P0, P1))$. So there is a greedy sequence of flips that leads to one or more satisfied

clauses than before, otherwise the solution would not be better.
By swapping x_i, it swaps either into the partition with $T0$ and turns from 0 to 1 in Positive-not-all-equal-max-3Sat/Kernighan-Lin, or it swaps into the partition not containing $T0$, which means x_i^{NAE} flips from 1 to 0. If x_i' is in the opposite partition of x_i, x_i' will be swapped with x_i. Then we do not lose the weight of $W+1$ of the edge between them. If x_i is in the same partition as x_i', then there is an x_k and x_k' in the opposite partition, because the partitions have the same size. By swapping x_i with x_k or x_k', we gain the weight of the clause containing x_i, the weight $W+1$ of the edge between x_k and x_k' and in some cases we gain or lose the weight of a clause containing x_k^{NAE} if such a clause exists. If x_k has already an edge to the other partition that corresponds to a clause, we will swap x_i with x_k' instead, so we do not lose that weight. But whatever we do, by swapping some nodes we gain a solution with better costs than $(P0, P1)$. Therefore $(P0, P1)$ was not optimal. [Yan88], [JPY88]

The functions f and g can be computed in polynomial time.

4.5. Max-Cut

Definition 4.5 Let $G = (V, E)$ be a graph with weight $w(e) \in \mathbb{N}^+$ for all edges $e \in E$. A solution is any partition $(P0, P1)$ of V and the cost $c(P0, P1)$ is the sum of the weights of the edges between $P0$ and $P1$, so the sum of the edges that have one endpoint in $P0$ and one endpoint in $P1$. The optimal solution is a solution with maximum cost $c(P0, P1)$.
This problem is quite similar to the Max-Uniform-Graph-Partitioning, with the difference that here the partitions do not need to have the same size.

Flip A partition $(P2, P3)$ is a neighbor of $(P0, P1)$ if we put one node from $P0$ to $P1$, or from $P1$ to $P0$. [MAK07]

Kernighan-Lin A partition $(P2, P3)$ is a neighbor of $(P0, P1)$ if $(P2, P3)$ can be obtained by a greedy sequence of flips, where the most profitable or the least unprofitable node is put from $P0$ to $P1$ or vice versa. Note that no node is allowed to be flipped twice.

Theorem 4.10 *Max-Cut/Flip is PLS-complete.*

Proof. Max-Cut/Flip is in PLS.

There is a valid PLS-reduction (f, g) from Positive-not-all-equal-max-3Sat/Flip, which is PLS-complete, to Max-Cut/Flip.

The proof is quite similar to the Max-Uniform-Graph-Partitioning reduction, as the problem is the same with the difference, that here the partitions do not need to have

the same size. It was first proven by [SY91] and we will have a more detailed view on the proof now.

The instance I^{NAE} of Positive-not-all-equal-max-3Sat consists of x_1^{NAE}, ..., x_n^{NAE} binary variables, and a set C of clauses. Each clause $c \in C$ has a weight $w(c)$ which is a positive integer.

The function f of the reduction constructs a graph $G = (V, E)$ of I^{NAE} in the following way:

Let there be one node x_i for each variable x_i^{NAE} in I^{NAE} and two nodes T and F for the constants 1 and 0. In this reduction we do not need the second node x_i' for each x_i as in the Max-Uniform-Graph-Partitioning reduction, because here the partitions do not need to have the same size.

The edges are built in the same way as in the Max-Uniform-Graph-Partition reduction:

- There is an edge between T and F with weight $W+1$, where W is the total weight of all clauses in C.
- For each clause $c =$NAE(x_p^{NAE}, x_q^{NAE}) there is an edge between the node x_p and x_q with weight $w(c)$.
- For each clause $c =$NAE$(x_p^{NAE}, x_q^{NAE}, x_r^{NAE})$ there are three edges, between x_p and x_q, x_q and x_r, and x_p and x_r. All of them have weight $\frac{w(c)}{2}$.
- If there is a constant 1 or a 0 in a clause, the corresponding node is T or F.
- If there are several edges between two nodes x_p and x_q, there is written only one edge with the sum of the weights of all edges between them.

The function g transforms a solution $(P0, P1)$ of $f(I^{NAE})$ to an assignment of x_1^{NAE}, ..., x_n^{NAE} in the following way: $x_i^{NAE} = 1$ if $x_i \in Pk$ and $T \in Pk$, otherwise $x_i^{NAE} = 0$.

In a locally optimal partition T and F will be in opposite partitions, as having them in the same partition can never be as good as having them in different partitions. Having them in the same partition, the highest possible weight to gain is W, but we gain $W+1$ if we put T and F into different partitions. Two nodes being in different partitions means that the corresponding variables have a different value. If they have an edge between them, so if their corresponding variables in I^{NAE} are in the same clause c, we gain the weight $w(c)$ of the clause, and respectively the weight $w(c)$ of the edge. So maximizing the weight of the clause leads to the same corresponding solutions as maximizing the weight of the edges between the partitions, and therefore to the same local and global optima. [Yan88], [SY91]

The functions f and g can be computed in polynomial time.

This reduction is tight, as proven in [SY91, p. 64].

Claim 4.3 *Max-Cut/Keringhan-Lin is PLS-complete.*

This Theorem has been stated without proof in [MAK07]. [MAK07] also claims that PLS-completeness can be proven with the tight PLS-reduction from Positive-not-all-equal-max-3Sat/Keringhan-Lin to Max-Cut/Keringhan-Lin, where the reduction is identical to the one from Positive-not-all-equal-max-3Sat/Flip to Max-Cut/Flip. The construction of the reduction is the same as in the proof of Theorem 4.10 on page 41, even though there is a difference in the neighborhood structure. If s is a local optimum in $f(I^{NAE})$, then $g(I^{NAE}, s)$ is a local optimum in I^{NAE}.

4.6. Min-Independent-Dominating-Set-B

Definition 4.6 An instance I consists of a graph $G = (V, E)$ with weights $w(v)$ for each vertex $v \in V$. The degree of each vertex v is bounded by a constant $B \in \mathbb{N}^+$.
A solution s of I is a subset of all nodes $s \subseteq V$ with the following characteristics:

- if a node is in the set s, none of the adjacent nodes is allowed to be in s
- if a node is not in the set s, at least one of the adjacent nodes has to be in s

The cost of a solution is the sum of the weights of the nodes in the set.
We want to minimize the cost.

k-Flip A neighbor r of a solution set s is obtained by flipping at most k nodes into or out of the set.

Theorem 4.11 *Min-Independent-Dominating-Set-B/k-Flip is PLS-complete.*

Proof. This has been proven by [Kla96, Lemma 2.3] via a tight PLS-reduction from Min-4Sat-B'/Flip to Min-Independent-Dominating-Set-B/k-Flip. Min-Independent-Dominating-Set-B/k-Flip is in PLS.

4.7. Weighted-Independent-Set

Definition 4.7 Let $G = (V, E)$ be a graph with weight $w(v) \in \mathbb{N}^+$ for all nodes $v \in V$. A solution s is a subset of nodes which forms an independent set, so a set of nodes where each edge $e \in E$ is incident to at most one vertex in s. The cost of a solution is the sum of the nodes in the set. We want to maximize the cost.

Change A neighboring set r of a solution s is obtained by adding a vertex u to s, and removing all adjacent vertices of u in s. Then augment it to a maximum independent set, if it is not yet maximal.

Claim 4.4 *Weighted-Independent-Set/Change is PLS-complete.*

Weighted-Independent-Set/Change is in PLS. [JPY88] states that PLS-completeness can be proven by a similar proof as for Max-Uniform-Graph-Partitioning/Kernighan-Lin. [MAK07] states that [SY91] showed PLS-completeness via a PLS-reduction from Max-2Sat/Flip to Weighted-Independent-Set/Change, though [SY91] only states it can be shown by a NP-reduction from 3Sat or Sat.

4.8. Maximum-Weighted-Subgraph-with-property-P

Definition 4.8 Let $G = (V, E)$ be a graph with weight $w(v) \in \mathbb{N}^+$ on the nodes $v \in V$. A solution is a subgraph $s \subseteq V$ so that s satisfies property P. The property P must be hereditary, which means that if a graph satisfies P, then all subgraphs satisfy P too. Moreover, P has to be non-trivial, which means that infinitely many graphs satisfy P, and infinitely many violate P. P must be verifiable in polynomial time.
The cost of a solution is the sum of the weights of the nodes in the subgraph.
We want to maximize the cost.

Change Let a solution s be a subset of V. For each $v \in V$ there is a set W_v, which is a neighbor of s. W_v is determined by the following algorithm:
If the graph $(V, \emptyset)$ satisfies P, then W_v is the set s without the nodes that are adjacent to v.
Otherwise, W_v is the set s without all the nodes that are not adjacent to v.
Join W_v with v, if v is not already in the set, and if $W_v \cup v$ still satisfies P.
Then add all nodes $u \in V \setminus W_v$, ordered by weight, for which $W_v \cup u$ satisfies P. [Shi97]

With this problem and an appropriate property P we can model problems like Max-clique and Max-acyclic-subgraph. [Shi97]

Theorem 4.12 *Maximum-Weighted-Subgraph-with-property-P/Change is PLS-complete if property P = "has no edges".*

Proof. [Shi97] states that Max-Weighted-Subgraph-with-property-(P="has no edges")/ Change equals Weighted-Independent-Set/Change , and has been proven in Theorem 4.4 on the previous page.

Theorem 4.13 *Maximum-Weighted-Subgraph-with-property-P/Change is PLS-complete.*

Proof. [Shi97] proved PLS-completeness via a tight PLS-reduction from Weighted-Independent-Set/Change to Maximum-Weighted-Subgraph-with-property-P/Change. Maximum-Weighted-Subgraph-with-property-P/Change is in PLS.

Note that the PLS-completeness of Weighted-Independent-Set/Change is just a claim without proof.

4.9. Set-Cover

Definition 4.9 An instance I of Set-Cover consists of a basic set B of size $|B| = n$. There is a set of subsets of B called $C = \{C_1, C_2, ..., C_k\}$. Each subset has a weight $w : C \to \mathbb{N}$. A solution s is a subset of C, where $\bigcup_{c_i \in s} C_i = B$.
The cost of a solution is the sum of the weights of the sets $C_i \in s$.
We want to find a solution with minimum cost.

k-change A solution r is neighbor of a solution s if they differ in at most k elements C_i.

Theorem 4.14 *Set-Cover/k-change is PLS-complete for each $k \geq 2$.*

Proof. [DS10] proved PLS-completeness via a tight PLS-reduction from (3, 2, r)-Max-Constraint-Assignment/Change (See Section 4.21 on page 54) to Set-Cover/k-change. Set-Cover/k-change is in PLS.

Note that Set-Cover/k-change is polynomial time solvable for $k = 1$ [DS10].

4.10. Metric-Traveling-Salesman-Problem (Metric-TSP)

Definition 4.10 Let $G = (V, E)$ be a complete graph with weights $w(e)$ on the edges $e \in E$. If the weights fulfill the triangle inequality, the problem is called Metric-TSP. A solution T, also called a "tour", is a Hamilton circle, that passes each node exactly once. The cost of a solution is the length of the tour, which is the sum of the weights of the edges in the tour. The aim is to find the tour with the minimum cost.

k-change A solution r is neighbor of a tour T, if r can be obtained by replacing at most k edges. [Kre89] An example of a 2-change would be the exchange of the two edges (a, b) and (c, d) by the edges (a, c) and (b, d). [Yan88]

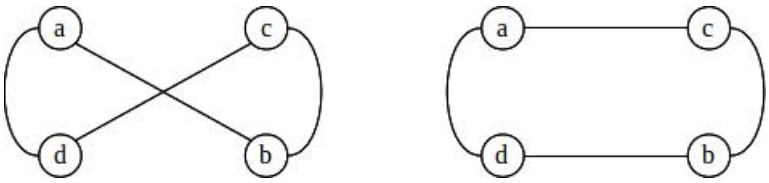


Figure 11: Example of the 2-change neighborhood [Yan88, Figure 2.1]

Lin-Kernighan A rotation: Given a tour, if we remove one edge (a, b), then we obtain a Hamilton path with endpoints a and b. Now we add the cheapest edge (b, c) to the tour. Then there is exactly one edge (c, d) which is deleted to make the tour a Hamilton path again.

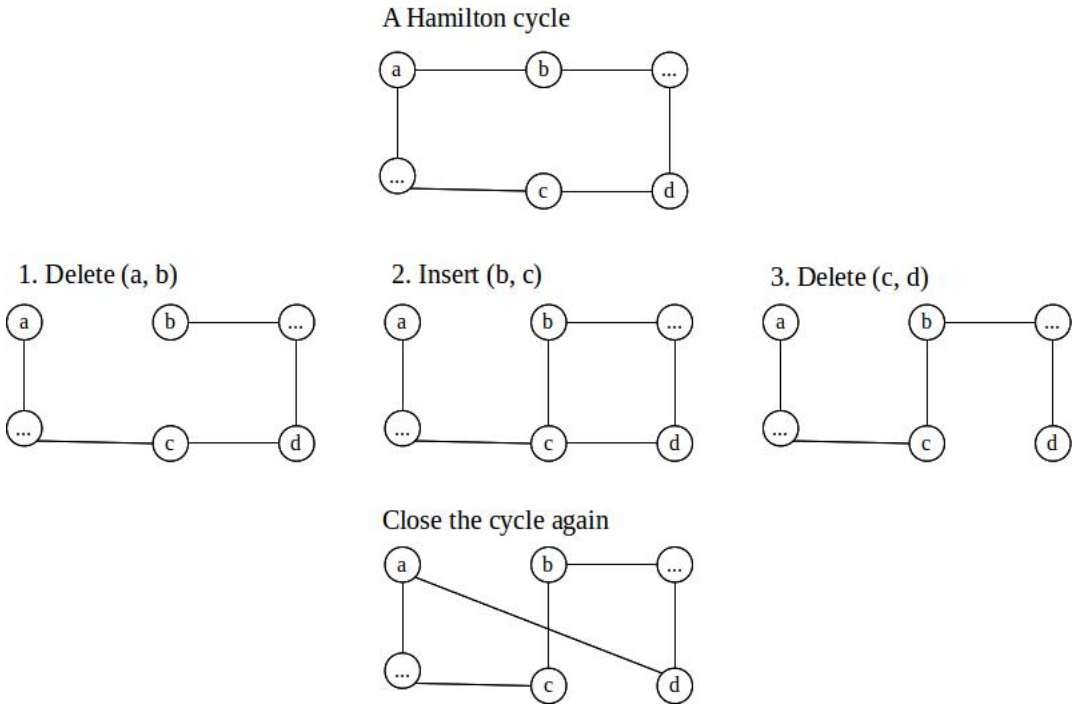


Figure 12: A rotation

Lin-Kernighan is a sequence of greedy rotations. At the end it closes the path to a cycle again [Yan88]. More mathematically, the Lin-Kernighan neighborhood function does the following steps to find the best neighbor of a tour T:

1. All 2-change and 3-change neighbors of T are included in $N(I, T)$.
2. Delete an edge (a, b) so there is a Hamilton path $H_1 = T \setminus (a, b)$.
3. To build H_i (the first time we do this step this is H_2), we do a rotation. Consider all edges that are not marked yet, and that are incident to the endpoint b in the Hamilton path H_{i-1}. If there is an edge (b, c) that leads to a better Hamilton cycle, when we delete (c, d) and add (d, a), this is the next neighboring tour $T_i = H_{i-1} \cup (b, c) \cup (d, a) \setminus (c, d)$. $H_i = H_{i-1} \cup (b, c) \setminus (c, d)$ where d is our new endpoint. The edges (b, c) and (c, d) are marked, they are not allowed to be used again.
   Note that if there are more than one edge which leads to a better Hamilton cycle, the one with the highest cost reduction is chosen.
   Then Step 3 is repeated. If there isn't an edge that leads to a better cycle, we add

the edge between the endpoints of the Hamilton path and terminate.
Otherwise the repetition stops when all edges are marked.

4. We have the neighbors $N(I,T) = \{T_1, T_2, ..., T_k\}$ from which we choose the best one to continue.

[Yan88], [MAK07]

[PSY90] does an extra step: If there is no better neighbor after step 4, choose a different edge (a', b') to delete at the beginning and repeat the algorithm from step 2.

This is not the Lin-Kernighan heuristic Lin and Kernighan actually defined in [LK73], as an edge is here allowed to enter the tour and later depart again, which is not the case in the original definition. [Yan88]

Theorem 4.15 *Metric-TSP/k-Change is PLS-complete.*

Proof. [Kre89] proved PLS-completenes via a PLS-reduction from Max-4Sat-B/Flip to Metric-TSP/k-Change. Note that [Kre89] calls Max-4Sat-B/Flip "CNF-SAT". Metric-TSP/k-Change is in PLS.

Theorem 4.16 *Metric-TSP/Lin-Kernighan is PLS-complete.*

Proof. [PSY90] proved PLS-completeness via a tight PLS-reduction from Max-2Sat/Flip to Metric-TSP/Lin-Kernighan. Metric-TSP/Lin-Kernighan is in PLS.

4.11. Local-Multi-Processor-Scheduling

Definition 4.11 An instance I consists of a number of $m \in \mathbb{N}$ identical machines and a set of Jobs $J = \{j_1, ..., j_n\}$ where every job has a processing time $w : J \to \mathbb{N}_0$. A solution s is a function, which is the assignment of all jobs to the machines.
The cost c of a solution is a tuple consisting of the makespan, which is the maximum time we need for the assignment to finish all processes, and the number of machines that need exactly that makespan time. The aim is to find an assignment with the minimum makespan q and the minimum number of makespan machines for that q.

k-change The neighbors of a solution assignment s are all assignments r, where up to k jobs are relocated. The neighbor r has better costs than s, if the cost tuple of r is lexicographically smaller than the cost tuple of s.

Theorem 4.17 *Local-Multi-Processor-Scheduling/k-change is PLS-complete.*

Proof. [DMT09] proved this via a tight PLS-reduction from Weighted-3Dimensional-Matching/(p, q)-Swap (See Section 4.22 on page 55) to Local-Multi-Processor-Scheduling/(2p+q)-change, where $(2p + q) \geq 8$. Local-Multi-Processor-Scheduling/k-change is in PLS.

4.12. Selfish-Multi-Processor-Scheduling

Definition 4.12 An instance I consists of a number of $m \in \mathbb{N}$ identical machines and a set of Jobs $J = \{j_1, ..., j_n\}$ where every job has a processing time $w : J \to \mathbb{N}_0$. A solution s is a function, which is the assignment of all jobs to the machines.
The cost c of a job on machine m_i is the sum of the processing time of all jobs that are assigned to m_i.
The cost of a solution is an n-vector with the costs of all jobs. The aim is to find a solution with the lexicographically smallest cost vector.

k-change-with-property-t A "coalition" is a subset of J. The cost of a coalition is the maximum cost of one of its members. There can be any number of coalitions.
The neighbors of a solution assignment s are all assignments r, where up to k jobs are relocated, with the restriction that all coalitions in this relocation have strictly better costs than in s. The neighbor r has better costs than s, if the cost vector of r is lexicographically smaller than the cost vector of s.

Theorem 4.18 *Selfish-Multi-Processor-Scheduling/k-change-with-property-t is PLS-complete.*

Proof. [DMT09] proved this via a tight PLS-reduction from Weighted-3Dimensional-Matching/(p, q)-Swap (See Section 4.22 on page 55) to (2p+q)-Selfish-Multi-Processor-Scheduling/k-change-with-property-t, where $(2p + q) \geq 8$.
Selfish-Multi-Processor-Scheduling/k-change-with-property-t is in PLS.

4.13. General-Congestion-Games

Definition 4.13 The input of a General-Congestion-Game is a set N of n players, a finite set R of resources and a set of actions $S_i \in Powerset(R)$ for each player i, which contains the resources player i is allowed to choose from. Furthermore there is a delay function $d : R \times \{1, ..., n\} \to \mathbb{Z}$. $d(r, j)$ is nondecreasing in j. It defines the price for each resource depending on how many players choose this resource. An action combination, or a state is $s = (s_1, ..., s_n)$, where $s_i \subseteq S_i$ are the resources player i actually chose. For any combination s we define $f_s(r) = |\{i | r \in s_i\}|$ as a congestion of a resource r, so the number of players that choose resource r in state s. $u_i(s) = -\sum_{r \in s_i} d(r, f_s(r))$ is the payoff function of player i. It returns the price player i has to pay for all resources r he chose in state s.

The aim is to find an action combination s for the congestion game that minimizes the following cost function: [MAK07, p.206]

$$\sum_{r \in R} \sum_{j=1}^{f_s(r)} d(r, j)$$

Change A solution state p is neighbor of a state s if p can be obtained from s by changing the action of one player.

Pure Nash Equilibrium A pure Nash Equilibrium is a state s, where for each player i changing the action of this player i does not improve the payoff of that player. The local optima of General-Congestion-Games/Change are pure Nash Equilibria. Rosenthal [Ros73] proved that there is always a pure Nash Equilibrium.

A General-Congestion-Game is symmetric, if each player has the same action set.

Theorem 4.19 *Finding the pure Nash Equilibrium in a General-Congestion-Game/Change is PLS-complete.*

Proof. [FPT04] proved PLS-completeness via a tight PLS-reduction from Positive-not-all-equal-max-3Sat/Flip to General-Congestion-Game/Change.
General-Congestion-Game/Change is in PLS.

Theorem 4.20 *Finding the pure Nash Equilibrium in a Symmetric General-Congestion-Game/Change is PLS-complete.*

Proof. Even with this restriction of symmetry [FPT04] proved PLS-completeness via a tight PLS-reduction from an asymmetric General-Congestion-Game/Change to symmetric General-Congestion-Game/Change. Symmetric General-Congestion-Game/Change is in PLS.

4.14. Network-Congestion-Games

Definition 4.14 A Network-Congestion-Game is a variant of a General-Congestion-Game (See Section 4.13 on the previous page) with the functions $d(s, j)$, $f_s(r)$ and $u_i(s)$ defined as before. Let $G = (V, E)$ be a directed graph where there are two nodes a_i, $b_i \in V$ for each player i and the edges are the resources. Each player i searches for a shortest path from a_i to b_i. It is a General-Congestion-Game where the set of actions is restricted by the possible paths from a_i to b_i. A solution is a possible action combination s, which represents a path from a_i to b_i for each player i.

Change A solution state p is neighbor of a state s if p can be obtained from s by changing the action of one player.

A Network-Congestion-Game is called symmetric, if all players have the same origin and destination and asymmetric if not.

Here, as well as in General-Congestion-Games, a local optimum corresponds with a pure Nash Equilibrium.

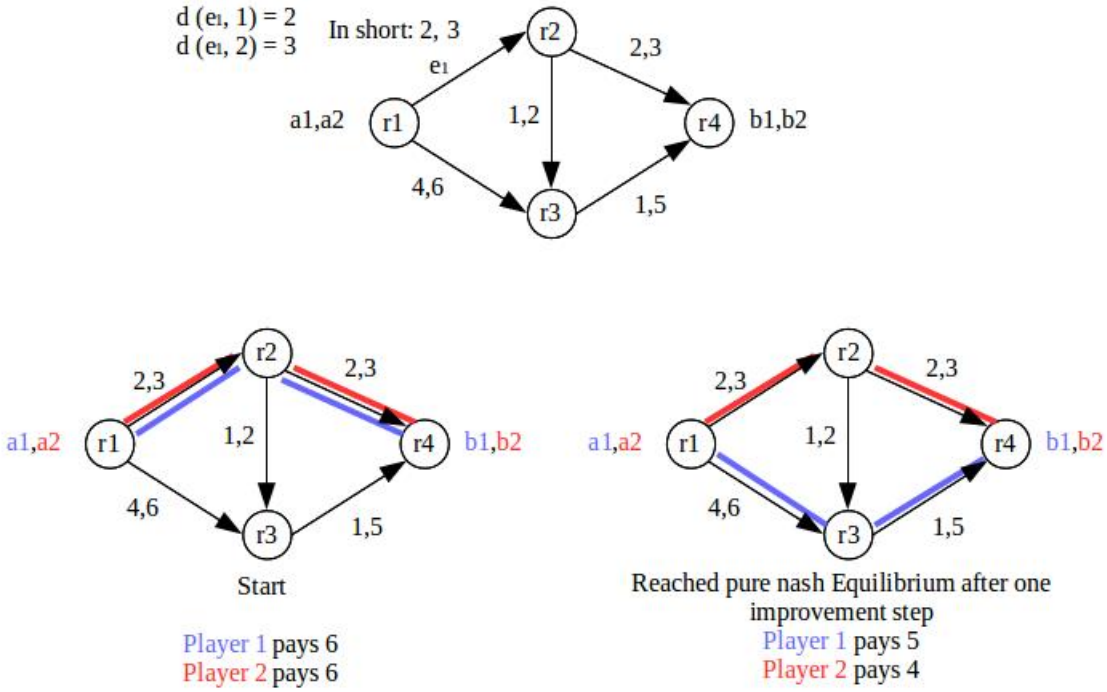


Figure 13: Example of a Network-Congestion-Game with two players, player 1 with startpoint a_1 and endpoint b_1, and player 2 with startpoint a_2 and endpoint b_2.

Theorem 4.21 *Asymmetric Directed-Network-Congestion-Games/Change is PLS-complete.*

Proof. [FPT04] proved PLS-completeness via a tight reduction from Positive-not-all-equal-max-3Sat/Flip to Directed-Network-Congestion-Games/Change in 2004. [ARV08, Theorem 12] proved the same in 2008 via an easier tight PLS-reduction from 2-Threshold-Games/Change to Directed-Network-Congestion-Games/Change. Asymmetric Directed-Network-Congestion-Games/Change is in PLS.

Theorem 4.22 *Asymmetric Undirected-Network-Congestion-Games/Change is PLS - complete.*

Proof. [ARV08, Theorem 13] proved PLS-completeness via a tight PLS-reduction from 2-Threshold-Games/Change (See Section 4.15 on the following page) to Asymmetric Undirected-Network-Congestion-Games/Change. Asymmetric Undirected-Network-Congestion-Games/Change is in PLS.

[FPT04] and [ARV08] state that there is a polynomial time algorithm for symmetric Network-Congestion-Games. Even though one can simulate an asymmetric Network-Congestion-Game by a symmetric one, the asymmetric Network-Congestion-Games are still PLS-complete.

4.15. Threshold-Games

Definition 4.15 A Threshold-Game is a special case of a General-Congestion-Game as defined in Section 4.13 on page 48. The set of resources is divided into two disjoint subsets R_{in} and R_{out} with $|R_{out}| = |N|$, where N is the set of players. Each player i only has two strategies, $S_i^{out} = \{r_i\}$ for a unique resource $r_i \in R_{out}$, where no two players are interested in the same resource r_i, and a strategy $S_i^{in} \subseteq R_{in}$. A resource r_i has a fixed cost T_i called the threshold of player i. Player i prefers to choose strategy S_i^{in} against strategy S_i^{out} if the cost of S_i^{in} is smaller than the cost of S_i^{out} . A Threshold-Game is called 2-Threshold-Game, if R_{in} contains one resource $r_{\{i,j\}}$ for every unordered pair of players $\{i, j\} \subset N$. Each player i has a strategy $S_i^{in} = \{r_{\{i,j\}} | j \in N, j \neq i\}$. It follows that exactly two players are interested in resource $r_{\{i,j\}}$. A solution s is an action combination.

Change A solution state p is neighbor of a state s if p can be obtained from s by changing the action of one player.

Theorem 4.23 *Finding a pure Nash Equilibrium in a 2-Threshold-Game/Change is PLS-complete.*

Proof. [ARV08] proved 2-Threshold-Games/Change PLS-complete via a tight reduction from Max-Cut/Flip to 2-Threshold-Games/Change. 2-Threshold-Games/Change is in PLS.

4.16. Market-Sharing-Games

Definition 4.16 A Market-Sharing-Game is a General-Congestion-Game(Section 4.13 on page 48) with a set N of the n players and a set R of m resources, or markets. One can view it as a bipartite graph $G = (N \cup R, E)$ where an edge between a player and a market indicates that a player is interested in that market. Each market r has a cost c_r and a profit $q_r \in \mathbb{N}$ which is equally distributed between the players who have allocated this market. The delay function $d(r, f_s(r))$ for each market r is here defined as $d(r, f_s(r)) = \frac{q_r}{f_s(r)}$, where $f_s(r)$ is again the number of players who allocated market r. In contrast to a general congestion game, each player has a budget B_i which is the limit of what he can pay. The budget implicitly defines which strategies each player has. The sum of the costs c_r of the markets a player picks has to be smaller than his budget. The players are interested in allocating markets with the maximum delay, as it is the profit they get, instead of the minimum delay as in general congestion games. This can be achieved by considering payoffs to be negative delays.

Change A solution state p is neighbor of a state s if p can be obtained from s by changing the action of one player.

Theorem 4.24 *Finding a pure Nash Equilibrium in Market-Sharing-Game/Change with polynomial bounded costs is PLS-complete.*

Proof. [ARV08] proved PLS-completeness via a tight PLS-reduction from 2-Threshold-Games/Change to Market-Sharing-Game/Change. Market-Sharing-Game/Change is in PLS.

4.17. Overlay-Network-Design

Definition 4.17 Let $G = (V, E)$ be an undirected graph with weight $w(e)$ on the edges $e \in E$. Consider the complete graph $G' = (V, E')$ of G. Weight $w'(e') = w(e)$ if $e' = (v1, v2) \in E$, and the weight of the shortest path from v_1 to v_2 otherwise. There are N players. A solution is a spanning tree in G' for each of the N players. The cost of a solution is the sum of the weights on the edges in the tree. The aim is to find a minimum spanning tree for each player.

Change A solution tree p is neighbor of a tree s if p can be obtained from s by changing the action of one player, which means adding an edge to the tree, and removing the edge with the highest weight in the unique circle.

Theorem 4.25 *Finding a pure Nash Equilibrium in an Overlay-Network-Design/Change is PLS-complete.*

Proof. [ARV08, Theorem 18] proved PLS-completeness via a reduction from 2-Threshold-Games/Change to Overlay-Network-Design/Change. Overlay-Network-Design/Change is in PLS. Analogously to the proof of asymmetric Directed-Network-Congestion-Game/ Change [ARV08, Theorem 12], the reduction is tight.

4.18. Stable Configuration in a Hopfield Network

Definition 4.18 Let $G = (V, E)$ be an undirected graph with weights $w(e)$ for each edge $e \in E$ and a threshold $t(v)$ for each node $v \in V$. Weights and thresholds can be positive or negative. A solution, also called a configuration, is the assignment of a state $s_v \in \{1, -1\}$ to each node $v \in V$. A node v is happy if the majority of its neighbors has the same state as the node itself. More mathematically, a node is happy if:

- $s_v = 1$ and $t(v) + \sum_{(u,v)\in E} w((u, v))s_u \geq 0$
- or $s_v = -1$ and $t(v) + \sum_{(u,v)\in E} w((u, v))s_u \leq 0$

A configuration is stable if all nodes are happy. Mathematically this corresponds to the local maxima of the following function:
$\sum_{(u,v)\in E} w((u,v))s_u s_v + \sum_{v\in V} s_v t(v)$ [SY91]

Flip A solution r is a neighbor of a solution s, if s and r differ in the state of one node. So we obtain r, if we change the state of one node in s. [SY91]

Theorem 4.26 *Stable-Configuration/Flip is PLS-complete if the thresholds are 0 and the weights are negative.*

Proof. [Yan88, p.41], [SY91] and [PSY90] prove PLS-completeness for a Stable-Configuration/Flip where the thresholds are 0 and the weights on the edges are negative, via a tight PLS-reduction from Max-Cut/Flip to Stable-Configuration/Flip. Stable-Configuration/Flip is in PLS.

4.19. Nearest-Colorful-Polytope

Definition 4.19 An instance I is a set of families $P = \{P_1, ..., P_n\}$, where each $P_i \subset \mathbb{R}^d, d \in \mathbb{N}$ is a color. Informally speaking we can say i is a color and P_i is the set of all points with this color.
A solution s is a perfect colorful choice, so a set of one point of each P_i.
$conv(s)$ is the convex hull of a pointset s.
The cost $c(s)$ of a solution is the smallest distance from a point in $conv(s)$ to the origin:
$c(s) = |conv(s)|_1 = min\{||q||_1 | q \in conv(s)\}$
We want to minimize the cost.

Change A neighbor r of a solution s is the colorful choice obtained from s by exchanging one point with another point of the same color.

Theorem 4.27 *Nearest-Colorful-Polytope/Change is PLS-complete.*

Proof. [MS14] proved PLS-completeness via a PLS-reduction from Max-2Sat/Flip to Nearest-Colorful-Polytope/Change. Nearest-Colorful-Polytope/Change is in PLS.

4.20. Min/Max-0-1-Integer-Programming

Definition 4.20 We want to minimize (or maximize) the function $C(x, A, b, c) = c^T x$ where $c \in \mathbb{N}^n$ and $x \in \{0,1\}^n$ under the constraint that $Ax \geq b$ where $A \in \mathbb{Z}^{m\times n}$ and $b \in \mathbb{Z}^m$. An instance consists of A, b and c. A solution is an assignment of x so that the constraint is satisfied. The cost of a solution is what $C(x, A, b, c)$ returns.

k-Flip A solution r is a neighbor of solution s if the Hamming distance H between s and r is at most k, so $H(s,r) \leq k$.

Theorem 4.28 *Min-0-1-Integer Programming/k-Flip is PLS-complete.*

Proof. This has been proven by [Kla96, Lemma 2.4] via a tight PLS-reduction from Min-4Sat-B'/Flip to Min-0-1-Integer Programming/k-Flip. Min-0-1-Integer Programming/k-Flip is in PLS.

Theorem 4.29 *Max-0-1-Integer Programming/k-Flip is PLS-complete.*

Proof. [Kla96] states that Min-0-1-Integer Programming/k-Flip can be PLS-reduced to Max-0-1-Integer Programming/k-Flip, but the proof is left out. Max-0-1-Integer Programming/k-Flip is in PLS.

4.21. (p, q, r)-Max-Constraint-Assignment

Definition 4.21 An instance I consists of a set of constraints C and a set of variables $X = \{x_1, ..., x_n\}$.
A constraint is a function that takes at most p input variables of X and returns a weight. Every variable x_i appears at most in q constraints. A solution s is an assignment of all the x_i to a value of $\{1, ..., r\}$.
The cost of a solution s is the sum of the weights the constraints return for the assignment s. We want to maximize this sum.

Change A neighbor r of a solution $s = (x_1, ..., x_n)$ is obtained by changing the value of one x_i. [DMT09]

Variant: (p, q, r)-max-constraint-assignment-k-partite A set of constraints is k-partite, if there exists a partitioning $t : X \to \{1, .., k\}$ of the variables so that for every two variables u and v that appear in the same constraint, $t(u) \neq t(v)$. [DM13, Definition 3]

Theorem 4.30 *(3, 2, 3)-Max-Constraint-Assignment-3-partite/Change is PLS-complete.*

Proof. [DM13] used the approach of [Kre89] to prove PLS-completeness via a tight PLS-reduction from Circuit/Flip to (3, 2, 3)-Max-Constraint-Assignment-3-partite/Change. (3, 2, 3)-Max-Constraint-Assignment-3-partite/Change is in PLS.

Theorem 4.31 *(2, 3, 6)-Max-Constraint-Assignment-2-partite/Change is PLS-complete.*

Proof. [DM13] proved PLS-completeness via a tight PLS-reduction from Circuit/Flip to (2, 3, 6)-Max-Constraint-Assignment-2-partite/Change. (2, 3, 6)-Max-Constraint-Assignment-2-partite/Change is in PLS.

Theorem 4.32 *(6, 2, 2)-Max-Constraint-Assignment/Change is PLS-complete.*

Proof. [DM13] proved PLS-completeness via a tight reduction from Circuit/Flip to (6, 2, 2)-Max-Constraint-Assignment/Change.
(6, 2, 2)-Max-Constraint-Assignment/Change is in PLS.

Theorem 4.33 *(4, 3, 3)-Max-Constraint-Assignment/Change is PLS-complete.*

Proof. (4, 3, 3)-Max-Constraint-Assignment/Change equals Max-4Sat-(B=3)/Flip (See Section 4.3 on page 36). [Kre89] proved PLS-completeness via a PLS-reduction from Max-circuit/Flip. [DM13] claims that the reduction can be extended so we obtain tightness. (4, 3, 3)-Max-Constraint-Assignment/Change is in PLS.

4.22. Weighted-3Dimensional-Matching

Definition 4.22 An instance I consists of a set $M \subset B \times G \times H$, with $|B| = |G| = |H| = n$ and a weight function, that assigns every triple (b, g, h), where $b \in B$, $g \in G$ and $h \in H$, a weight $w \in \mathbb{R}^+$.
A solution is a set of triples T, where $|T| = n$ and every element b, g or h is only in one triple.
The cost of a solution is the sum of the weights of the triples.
The aim is to find the solution with the highest cost.

(p, q)-Swap Restructuring at most p triples of a solution s, where at most q elements are changed, leads to a neighbor r.

Example 4.5 with neighborhood (2, 1)-Swap with n=3:

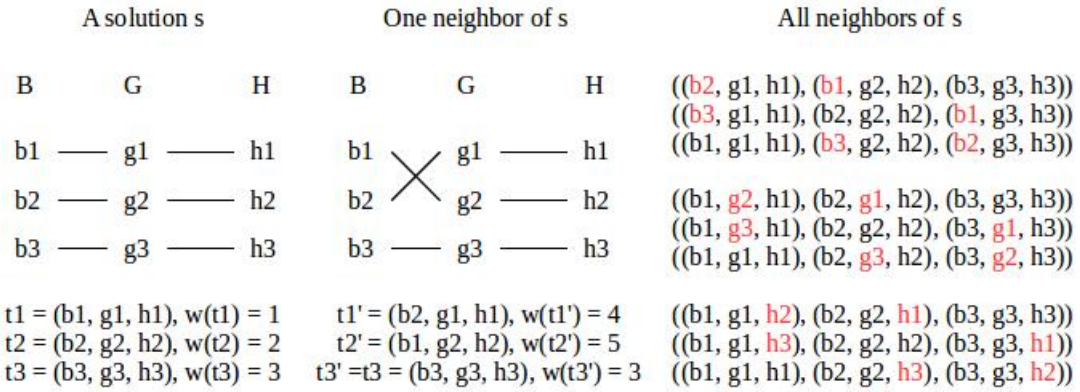


Figure 14: Example with neighborhood (2, 1)-Swap with n=3

Theorem 4.34 *Weighted-3Dimensional-Matching/(p, q)-Swap is PLS-complete for* $p \geq 9$ *and* $q \geq 15$.

Proof. [DMT09] proved this via a tight PLS-reduction from (2, 3, r)-Max-Constraint-Assignment-2-partite/Change to Weighted-3Dimensional-Matching/(p, q)-Swap. Weighted-3Dimensional-Matching/(p, q)-Swap is in PLS.

4.23. Other Problems

There are more problems proven to be PLS-complete:

In [DM13]

- (p, q, r)-Value-Constraint-Assignment

In [DS10]

- Exact-Cover-By-3-Sets/k-change, $k \geq 6$
- Set-Packing/k-change, $k \geq 2$
- Set-Splitting/k-change, $k \geq 1$
- Test-Set/k-change, $k \geq 1$
- Set-Basis/k-change, $k \geq 1$
- Hitting-Set/k-change, $k \geq 1$
- Intersection-Pattern/k-change, $k \geq 1$
- Comparative-Containment/k-change, $k \geq 1$

5. Conclusion

In the first Section we reviewed the definition of PLS and what is known about its relation to FP and FNP. The exact position of PLS with respect to FP and FNP is still an important open question.

Furthermore an introduction to PLS-reductions, tight PLS-reductions and PLS-completeness was given, and we proved that in the worst case it takes an exponential number of steps to reach a local optimum with the standard algorithm, which runs in pseudo-polynomial time. It follows that a local optimum can be found in polynomial time for a PLS-complete problem, if and only if local optimum can be found for all PLS-complete problems.

In Section 4 on page 21 we presented an overview of PLS-complete problems with a definition and a reference to the proof for each of them, with the aim to make future PLS-completeness proofs easier, as there are many problems one can reduce a new problem from.

There are some topics this work did not deal with, for example the question about the quality of a local optimum [MAK07, Chapter 5]. What exactly is the trade-off between a large neighborhood finding a good local optimum and a smaller neighborhood structure, finding a maybe not so good solution faster? There also are other possibilities to find a local optimum than with the standard algorithm, for example simulated annealing and tabu search [MAK07, Chapter 7].

Another issue is the standard local optimum problem: For a given problem, starting from a given initial solution, how can we establish the local optimum that the standard local search algorithm would have produced starting from this solution as quickly as possible? [ACZ14]
For all PLS problems, this is PSPACE-complete, which can be proven by using tight PLS-reductions. [Yan88, Chapter 5.4]

There are many more interesting questions regarding the complexity of local search. For example, how the complexity is influenced when we parallelize a local search algorithm. [Yan88, Chapter 6]

[JPY88] stated that a local search problem cannot be in PLS unless the subproblem of verifying local optimality is P-complete. This was proven wrong by Krentel, by proving Max-4-Sat-(B=3)/Flip to be PLS-complete, which can be verified in LOGSPACE [Kre89], [Kre90, Theorem 2.2].

All in all there are many possibilities to continue the research on this topic with the help of the basics presented in this work.

References

[LK73] S Lin and B.W. Kernighan. "An efficient heuristic algorithm for the traveling-salesman problem". In: *Operations Research* 21.2 (1973), pp. 498–516.

[Ros73] Robert W Rosenthal. "A class of games possessing pure-strategy Nash equilibria". In: *International Journal of Game Theory* 2.1 (1973), pp. 65–67.

[FM82] C.M. Fiduccia and R.M. Mattheyses. "A linear-time heuristic for improving network partitions". In: *Proceedings of the 19th ACM/IEEE Design Automation Conference* (1982), pp. 175–181.

[JPY88] David S Johnson, Christos H Papadimitriou, and Mihalis Yannakakis. "How easy is local search?" In: *Journal of computer and system sciences* 37.1 (1988), pp. 79–100.

[Yan88] Mihalis Yannakakis. "Computational complexity". In: *Local search in combinatorial optimization* (1988), pp. 19–55.

[Kre89] Mark W. Krentel. "Structure in locally optimal solutions". In: *Proceedings of the 30th Annual Symposium on Foundations of Computer Science* (1989), pp. 216–221.

[Kre90] Mark W Krentel. "On finding and verifying locally optimal solutions". In: *SIAM Journal on Computing* 19.4 (1990), pp. 742–749.

[PSY90] C.H. Papadimitriou, A. A. Schäffer, and M. Yannakakis. "On the complexity of local search". In: *Proceedings of the 22nd ACM Symposium on Theory of Computing* (1990), pp. 438–445.

[MP91] Nimrod Megiddo and Christos H Papadimitriou. "On total functions, existence theorems and computational complexity". In: *Theoretical Computer Science* 81.2 (1991), pp. 317–324.

[SY91] A. A. Schäffer and Mihalis Yannakakis. "Simple local search problems that are hard to solve". In: *SIAM journal on Computing* 20.1 (1991), pp. 56–87.

[Kla96] Hartmut Klauck. "On the hardness of global and local approximation". In: *Proceedings of the 5th Scandinavian Workshop on Algorithm Theory* (1996), pp. 88–99.

[Shi97] Shinichi Shimozono. "Finding optimal subgraphs by local search". In: *Theoretical Computer Science* 172.1 (1997), pp. 265 –271.

[FPT04] Alex Fabrikant, Christos Papadimitriou, and Kunal Talwar. "The complexity of pure Nash equilibra". In: *Proceedings of the thirty-sixth annual ACM symposium on Theory of computing* (2004), pp. 604–612.

[MAK07] Wil Michiels, Emile Aarts, and Jan Korst. *Theoretical aspects of local search.* Springer Science & Business Media, 2007.

[ARV08] Heiner Ackermann, Heiko Röglin, and Berthold Vöcking. "On the impact of combinatorial structure on congestion games". In: *Journal of the ACM (JACM)* 55.6 (2008), p. 25.

[Cor+09] Thomas H. Corman et al. *Introduction to algorithms*. Third edition. MIT press, 2009.

[DMT09] Dominic Dumrauf, Burkhard Monien, and Karsten Tiemann. "Multiprocessor scheduling is PLS-complete". In: *System Sciences, 2009. HICSS'09. 42nd Hawaii International Conference on* (2009), pp. 1–10.

[DS10] Dominic Dumrauf and Tim Süß. "On the Complexity of Local Search for Weighted Standard Set Problems". In: *CoRR* abs/1004.0871 (2010). URL: `http://arxiv.org/abs/1004.0871`.

[DM13] Dominic Dumrauf and Burkhard Monien. "On the PLS-complexity of maximum constraint assignment". In: *Theoretical Computer Science* 469 (2013), pp. 24–52.

[ACZ14] Eric Angel, Petros Christopoulos, and Vassilis Zissimopoulos. "Local Search: Complexity and Approximation". In: *Paradigms of Combinatorial Optimization: Problems and New Approaches, Volume 2* (2014), pp. 435–471.

[MS14] Wolfgang Mulzer and Yannik Stein. "Computational Aspects of the Colorful Carathéodory Theorem". In: *arXiv preprint arXiv:1412.3347* (2014).

A. Appendix

Preliminaries:

$|D| \geq 4$

$M := (3|D|)^{3|D|}$

$N_j := \frac{M}{(3|D|)^{2j}} = (3|D|)^{(3|D|)-2j}$

A.1.

$$\begin{aligned} \text{We shall prove: } M &> 3N_j \\ M &= (3 \mid D \mid)^{3|D|} \\ &= N_j \cdot (3 \mid D \mid)^{2j} \\ &= N_j \cdot \underbrace{3^{2j}}_{>3} \cdot \underbrace{\mid D \mid^{2j}}_{>1} \\ &> 3N_j \end{aligned}$$

A.2.

$$\text{We shall prove: } M - 3 \cdot N_j - 3^{|D|} > -M$$

$$\begin{aligned} & M - 3 \cdot N_j - 3^{|D|} > -M \\ \Leftrightarrow\ & 2M - 3 \cdot N_j - 3^{|D|} > 0 \end{aligned}$$

$$\begin{aligned} 2M - 3 \cdot N_j - 3^{|D|} &> 6N_j - 3N_j - 3^{|D|} && \text{as } M > 3N_j \text{ [A1]} \\ &= 3N_j - 3^{|D|} \\ &\geq 3 \cdot 3^{|D|} - 3^{|D|} && \text{as } N_j > 3^{|D|} \text{ [A3]} \\ &= 2 \cdot 3^{|D|} \\ &> 0 \end{aligned}$$

A.3.

$$
\begin{aligned}
\text{We shall prove: } N_j &> 3^{|D|} \\
N_j &\geq (3 \mid D \mid)^{|D|+2n} && \text{as [A9]} \\
&> (3 \mid D \mid)^{|D|} \\
&> 3^{|D|}
\end{aligned}
$$

A.4.

We shall prove: The maximum weight w of all clauses of the form $\{y_k, 0\}$ and $\{x'_{i1}, z_i\}$ is smaller than $3^{|D|}$

$$
\begin{aligned}
w &= (\sum_{k=1}^{m} 2^{k-1}) + 2n \\
&= 2^m - 1 + 2n \\
&< 2^m + 2n \\
&< 2^{|D|} + 2n && \text{as } m <\mid D \mid -n <\mid D \mid \\
&< 2^{|D|} + 2 \cdot 2^{|D|} && \text{as } m + n <\mid D \mid \text{ is } n <\mid D \mid -m <\mid D \mid < 2 \mid D \mid < 2^{|D|} \\
&= 2^{|D|} \cdot (1 + 2) \\
&= 3 \cdot 2^{|D|} \\
&< 3^{|D|} && \text{as } \mid D \mid \geq 4 \text{ by definition}
\end{aligned}
$$

A.5.

We shall prove: $(\mid D \mid -n) \cdot 3 \cdot N_1 < M - 2$

$$
\begin{aligned}
(\mid D \mid -n) \cdot 3 \cdot N_1 &= (\mid D \mid -n) \cdot 3 \cdot (3 \mid D \mid)^{3|D|-2} \\
&= (3 \mid D \mid -3n) \cdot (3 \mid D \mid)^{3|D|-2} \\
&= 3 \mid D \mid \cdot (3 \mid D \mid)^{3|D|-2} - (3 \mid D \mid)^{3|D|-2} \cdot 3n \\
&= (3 \mid D \mid)^{3|D|-1} - \underbrace{(3 \mid D \mid)^{3|D|-2} \cdot 3n}_{\substack{|D| \geq 4 \\ \Leftrightarrow 3|D|-2 \geq 10 \\ \Rightarrow (3|D|)^{3|D|-2} \geq 3^{10} > 1 \\ \Rightarrow 3n \cdot 3|D|^{3|D|-2} > 2}} \\
&< (3 \mid D \mid)^{3|D|-1} - 2 \\
&< (3 \mid D \mid)^{3|D|} - 2 \\
&= M - 2
\end{aligned}
$$

A.6.

We shall prove: $M - 3N_j + (3(\mid D \mid -n-j)N_{j+1}) - 3^{|D|} < M - 2$

$$
\begin{aligned}
M - 3N_j + (3(\mid D \mid -n-j)N_{j+1}) - 3^{|D|} &< M - 2 \\
\Leftrightarrow -3N_j + (3(\mid D \mid -n-j)N_{j+1}) - 3^{|D|} &< -2 \\
\Leftrightarrow (3(\mid D \mid -n-j)N_{j+1}) - 3^{|D|} &< 3N_j - 2
\end{aligned}
$$

$$
\begin{aligned}
\underbrace{(3(\mid D \mid -n-j)N_{j+1})}_{< \frac{1}{|D|} N_j [A7]} - 3^{|D|} &< \frac{1}{\mid D \mid} N_j - 3^{|D|} \\
&\leq \frac{1}{\mid D \mid} N_j - 3^4 && \text{as } \mid D \mid \geq 4 \\
&< \frac{1}{\mid D \mid} N_j - 2 \\
&\leq \frac{1}{4} N_j - 2 && \text{as } \mid D \mid \geq 4 \\
&< 3N_j - 2
\end{aligned}
$$

A.7.

$$\text{We shall prove: } (3(\mid D \mid -n-j)N_{j+1}) < \frac{1}{\mid D \mid} N_j$$

$$\begin{aligned}
(3(\mid D \mid -n-j)N_{j+1}) &= 3(\mid D \mid -n-j)(3 \mid D \mid)^{3|D|-2j-2} \\
&= 3(\mid D \mid -n-j) \cdot N_j \cdot (3 \mid D \mid)^{-2} \qquad \text{as [A8]} \\
&= (\frac{3 \mid D \mid}{(3 \mid D \mid)^2} - \underbrace{\frac{3n}{(3 \mid D \mid)^2}}_{>0} - \underbrace{\frac{3j}{(3 \mid D \mid)^2}}_{>0}) \cdot N_j \\
&\leq \frac{1}{3 \mid D \mid} \cdot N_j \\
&< \frac{1}{\mid D \mid} \cdot N_j
\end{aligned}$$

A.8.

$$\begin{aligned}
\text{We shall prove: } N_{j+1} &= (3 \mid D \mid)^{-2} \cdot N_j \\
N_{j+1} &= (3 \mid D \mid)^{3|D|-2j-2} \\
&= \underbrace{(3 \mid D \mid)^{3|D|-2j}}_{=N_j} \cdot (3 \mid D \mid)^{-2} \\
&= (3 \mid D \mid)^{-2} \cdot N_j
\end{aligned}$$

A.9.

N_j is smallest, if j is the highest value it can take. j is by definition $1 \leq j \leq \mid D \mid -n$ Therefore the smallest N_j is $N_{|D|-n}$, every other N_j is bigger.

$$\begin{aligned}
N_j &= \frac{(3 \mid D \mid)^{3|D|}}{(3 \mid D \mid)^{2j}} \\
N_j &\geq N_{|D|-n} \\
&= \frac{(3 \mid D \mid)^{3|D|}}{(3 \mid D \mid)^{2(|D|-n)}} \\
&= (3 \mid D \mid)^{|D|)+2n}
\end{aligned}$$