An Overview of What We Can and Cannot Do with Local Search

Petros Christopoulos*, Vassilis Zissimopoulos*

Résumé

Etant donné qu'on ne connaît pas un algorithme efficace pour résoudre les problèmes d'optimisation NP-difficiles, on développe plusieurs algorithmes approchés. Parmi ces algorithmes est la Recherche Locale qui est une méthode générale. On considère une structure de voisinage de solutions d'un problème d'optimisation et au lieu de trouver la meilleure solution dans le domaine, nous trouvons une solution, appelée optimum local, qui est la meilleure dans ce voisinage. Ainsi, l'heuristique standard de la recherche locale commence par une solution initiale et se déplace à une meilleure solution voisine pour aboutir à un optimum local. Cette simple méthode se démontre en pratique très performante produisant des solutions de bonne qualité dans de temps d'exécution raisonnable.

Le but principal de ce travail est de faire une synthèse du travail théorique qui est réalisé sur les limites de la recherche locale en général et son efficacité d'approximation pour de problèmes spécifiques. Ainsi, d'un côté nous présentons la théorie de PLS-complétude et nous montrons que pour un problème PLS-complet l'heuristique de la recherche locale standard nécessite dans le pire de cas de temps exponentiel. Nous montrons aussi que s'il est NP-difficile d'obtenir une $\varepsilon-$approximation d'un problème d'optimisation alors il n'existe pas de voisinage qui conduit à un optimum local $\varepsilon-$proche d'un optimum global, sauf si NP=co-NP. De l'autre côté, nous présentons plusieurs exemples de problèmes NP-difficiles pour lesquels certains voisinages peuvent garantir des optima locaux $\varepsilon-$proche d'un optimum global. Cette garantie est souvent, la meilleure qu'on puisse obtenir pour certains problèmes par n'importe quel algorithme. L'algorithme de la recherche locale est pseudopolynomial. Par conséquent, lorsque les problèmes sont sans poids ou avec des poids polynomialement bornés l'algorithme atteint un optimum local en temps polynomial.

Mots-clefs : recherche locale, PLS-complétude, approximation

*Department of Informatics and Telecommunications, University of Athens, 15784 Athens, Greece. {p.christopoulos, vassilis}@di.uoa.gr

Résumé

Since we do not know any algorithm to efficiently solve the NP-hard optimization problems, a lot of approximation algorithms have been evolved. A general method for this purpose is Local Search. One assumes a neighboring structure between the solutions of an optimization problem and wants to find a solution that is the best in its neighborhood, called a local optimum, instead of the best solution in the domain. So, the standard local search heuristic starts from an initial solution and keeps moving to some better neighbor until it reaches a local optimum. This simple method turns out to be very successful in practice both in its running time performance and on the quality of the solutions that produces.

The main purpose of this work is to sum up the theoretical work that has been done concerning the limits of local search in general and its proven approximation efficiency for particular problems. Hence, on the one hand we present the PLS-completeness theory and show that for the PLS-complete problems the standard local search heuristic takes exponential time in the worst case. We also show that if it is NP-hard to ε-approximate an optimization problem then there is no neighborhood which produces local optima only ε-close to global optima, unless NP=co-NP. On the other hand, we present numerous of examples of NP-hard optimization problems that under appropriate neighborhoods guarantee local optima ε-close to the global optima. Such guarantees are, in many cases, between the best ones for these problems by any algorithm. Local search heuristic is pseudopolynomial, so when the problems are unweighted or with polynomially bounded weights it reaches a local optimum in polynomial time.

Key words : local search, PLS-completeness, approximability

1 Introduction

Local Search is a widely used general method to approximately solve NP-hard optimization problems. Roughly speaking, an optimization problem has a solution set and a cost function, which assigns a value to each solution. Our aim is to find an optimal solution, that is a solution with the smallest or greatest value. We can obtain a *local search heuristic* for an optimization problem by superimposing a neighborhood structure on the solutions, i.e. we define for each solution a set of neighboring solutions. The heuristic starts from an initial solution, which is constructed by another algorithm or is chosen randomly, and keeps moving to some better neighbor as long as there is one. So the heuristic stops when it reaches a *locally optimal solution*, i.e. one that does not have any better neighbor.

An important theory for local search has been developed, about both its complexity and its approximation ability. This theory will be our issue for the first part of this paper

(Sections 4 to 6) and more particularly the answers that have been given to the following questions:

- What is the complexity of the local search heuristic? (that is, how fast can we find a local optimum?)
- What is the quality of the solutions that a local search heuristic can give? (that is, how close to the global optimum can be the local optimum).

Generally, for many interesting problems, the complexity of finding a locally optimal solution remains open, that is we do not know whether it can be found in polynomial time or not, by any algorithm. The complexity class PLS (Polynomial-time Local Search), which is presented in Section 4, was defined by Johnson, Papadimitriou and Yannakakis, in [1], exactly for the purpose of grouping such problems, provided that their neighborhood is polynomially searchable. This is the least requisition which is satisfied by the neighborhoods used in the common local search heuristics. PLS class lies somewhere between P and NP. It has been shown that a lot of important local search problems, which we will describe in Section 3, are complete for PLS under a properly defined reduction, that is why PLS characterizes the complexity of the local search problems in the same sense that NP characterizes the complexity of the "hard" optimization problems. Furthermore, we can analyze the complexity of many popular local search heuristics with the aid of the PLS-completeness theory.

Hence, through this theory, in Section 5, a (negative) answer to the first question, that of complexity, is given for a lot of important local search problems (for example TSP/k-Opt, GP/Swap and Kernighan-Lin, Max-Cut/Flip, Max-2Sat/Flip, Stable configurations for neural networks). However, there are still "open" problems, such as TSP/2-Opt, for which we do not know whether they can be solved polynomially by the local search heuristic.

About quality, we will see in Section 6 a theorem restrictive for the approximability of the (globally) optimal solutions of NP-hard problems, with the local search method. Additionally, we will see that it is also impossible, given a PLS problem to guarantee to find solutions ε-close to a local optimum, in polynomial time, by the standard local search heuristic. However, we can find solutions that are $(1+\varepsilon)$ times smaller (resp. bigger), for minimization (resp. maximization) problems, from their neighbors by an FPTAS.

The planning of a good neighborhood and hence of a successful local search heuristic is mainly achieved by experimental technics, trying to keep a balance between the quality of the solution and the time needed to be found. In Section 7, we describe many NP-hard problems for which the local search heuristic provably gives solutions ε-close to their global optima, in polynomial time, using appropriate neighborhoods. We also, present some characteristic proofs of this kind.

The importance of local search is based on its simplicity and its surprising efficiency in "real-life" problems. Despite its main use with NP-hard problems, local search is also used in many other cases that we will refer in Section 8. Hence, there are polynomial problems for which local search heuristics are better, in practice, than other polynomial algorithms, such as Linear Programming and Maximum Matching. For these problems we consider neighborhoods which guarantee that their local optima coincide with the global optima. Another use of local search is as a tool in proofs of existence of an object. For example, see in subsection 3.6 the proof that there is always a stable configuration in neural networks of the Hopfield model. The theory of local search was recently applied in Game Theory giving some interesting first results. Finally, local search has a powerful extension, non-oblivious Local Search, and is also the base of the most metaheuristics such as Tabu Search and Simulated Annealing.

The following section sets up a formal framework for local search and provides some useful definitions and questions, for the rest of this work.

2 Framework

A *general computational problem* Π has a set D_Π of instances and for each instance $x \in D_\Pi$ there is a set $\mathcal{J}_\Pi(x)$ of corresponding possible feasible answers. An algorithm solves problem Π if on input $x \in D_\Pi$ it outputs a member y of $\mathcal{J}_\Pi(x)$ or in case that $\mathcal{J}_\Pi(x)$ is empty, it reports that there is no such y. Inputs (instances) and outputs are represented as strings upon a finite alphabet, which without loss of generality can be considered to be $\{0, 1\}$.

There are different types of computational problems depending on the size of their output sets $\mathcal{J}_\Pi(x)$. The most general kind is a *Search* problem where each $\mathcal{J}_\Pi(x)$ can be consisted of zero, one or more elements. If $\mathcal{J}_\Pi(x)$ is non-empty for $x \in D_\Pi$ the problem is called *Total*. If $|\mathcal{J}_\Pi(x)| \leq 1$ for every x, then it is called *Functional*. An important special case of *Total Functional* problems, i.e. those for which holds $|\mathcal{J}_\Pi(x)| = 1$ is the set of the *Decision* problems, where for every instance the unique answer can be either "Yes" (1) or "No" (0). For instance, does a given graph has a Hamiltonian cycle?

An instance x of an *Optimization* problem Π is a pair $(\mathcal{J}_\Pi(x), f_\Pi(i, x))$, where $\mathcal{J}_\Pi(x)$ is any feasible solution set and $f_\Pi(i, x)$ is a *cost function* for every $i \in \mathcal{J}_\Pi(x)$, i.e. a mapping $f : \mathcal{J} \to \mathbb{R}$. The problem is to find an $i^* \in \mathcal{J}$ such that $f(i^*, x) \leq f(y, x)$ for a minimization problem or $f(i^*, x) \geq f(y, x)$ for a maximization problem, for every $y \in \mathcal{J}$. Such a point i^* is called a *global optimal solution* of the specific instance. The cost function assigns a cost on every solution of the instance of the problem, which is usually a positive integer. An optimization problem is a set $\mathcal{I}$ of such instances. These problems are in fact a kind of search problems, since they turn up by applying a cost

function on the latter (there can, also, be more than one optimal solutions with, the same, optimal cost).

Optimization problems, now, are divided in two categories: those whose solutions are consisted of variables, which take discernible values and are called *Combinatorial* (for example Integer Linear Programming, Graph Partitioning) and those whose solutions are consisted of variables, which take continuous values and are called *Continuous* (for example, Linear Programming[1]).

Defining a neighborhood structure on the set of the solutions of a Combinatorial Optimization problem, we obtain a *Local Search* problem. The Local Search problem is this: Given an instance x, find a *locally optimal solution* $\hat{ß}$. That is, a solution which does not have any strictly better neighbor (smaller cost in minimization problems or greater cost in maximization problems). This is a Search problem, too, because an instance may have many locally optimal solutions.

Given an instance x, a set $\mathcal{N}_\Pi(i, x)$ of neighboring solutions is assigned on every solution $i \in \mathcal{J}_\Pi(x)$. The neighbors are determined each time from the current instance x and the current solution i. From a Combinatorial Optimization problem and different neighborhood functions arise different Local Search problems, for this reason we symbolize them as $OP/\mathcal{N}$.

Even though the term *neighborhood* implies that the neighboring solutions are "close" to each other in some sense, i.e. that we can obtain one solution from the other by a small perturbation, this need not generally be the case. The neighborhood function can be very complicated and it does not even need to be symmetric. It is possible for a solution i to have a neighboring solution j, when j does not have i for neighbor. For example, consider the *Graph Partitioning* problem. A very simple neighborhood is *Swap*, which is defined as follows: a partition (A, B) of the set of the nodes V into two equal parts has as neighbors all the partitions that are obtained by swapping one node from A with one node from B. A much more complex neighborhood is being searched by the *Kernighan-Lin* algorithm in [2]. This neighborhood is not symmetric and depends on the weights of the edges. Additionally, if a partition is locally optimal under the Kernighan-Lin neighborhood then it will be locally optimal under Swap, too. Hence, finding local optima under Kernighan-Lin is at least as difficult as it is under Swap. Surprisingly, as we will see later, the two problems are actually polynomially equivalent.

In general, the more powerful a neighborhood is the harder one can search it and find local optima, but probably these optima will have better quality. The most powerful neighborhood is the *exact* neighborhood. In an exact neighborhood local and global optima are the same, and hence the local search problem coincides with the optimization problem. Of course, in every optimization problem one can make local and global optima

[1]Linear Programming, in particular, can also be considered as a Combinatorial Optimization problem, since the solution that we are looking for belongs in the finite set of the vertices of a polytope.

to coincide by selecting sufficiently large neighborhoods. But the problem is that in such a case it will be difficult for one to search the neighborhood. That is, finding whether a solution is locally optimal and if not finding a better neighbor will take exponential time. Therefore, an essential consideration is that we must be able to search the neighborhood efficiently.

For the problems that we will discuss later the following framework is used: A General Local Search problem Π consists of a set of instances D_Π and for each instance $x \in D_\Pi$ there is a solution set $\mathcal{J}_\Pi(x)$, a cost function $f_\Pi(i,x), i \in \mathcal{J}_\Pi(x)$ and a neighborhood function $\mathcal{N}_\Pi(i,x), i \in \mathcal{J}_\Pi(x)$. The problem is as follows: Given an instance x, find a locally optimal solution. *Note that the use of any algorithm is allowed, not necessarily of a local search algorithm.*

Besides the local search problem itself we will also discuss about the corresponding heuristic. The *standard local search algorithm* does the following actions: starts from an initial solution (which is obtained either randomly or by another algorithm), moves to a better neighboring solution and repeats the previous step, until it reaches a solution $\hat{ß}$, which has no better neighbors and is called *local optimum*. Notice that the complexity of a local search problem can be different from that of its corresponding standard local search heuristic. Consider, for instance, Linear Programming which is a problem in P, but Simplex, the local search algorithm used to solve it, has a worst-time complexity exponential. The two issues that we are concerned about a local search heuristic is its complexity (Section 5) and the quality of the solution that it finds, i.e. how close to the global optimum are the local optima (Section 7).

Assuming that the complexity of one iteration is polynomial, in Section 5 we are looking for the complexity in the worst case (as function of the size of the instance, for the instances and all their initial solutions) of the standard local search algorithm with a specific pivoting rule. This problem is called *Running Time Problem*. The running time of the local search algorithm, whether it is polynomially bounded or not, depends mainly on the number of the iterations.

The *pivoting rule* is the rule that is used by the heuristic to choose to which specific better solution, among all the better neighboring solutions that a solution might have, to move. In the examination of the local search algorithms we want to analyze the complexity for different pivoting rules and to find the best one.

Notice, also, that the standard local search algorithm is pseudopolynomial. This means that its complexity in the worst case depends from the numbers (weights, costs) that exist in the instance. If the number of the different solution costs is polynomial, then there will be, at most, a polynomial number of iterations and the algorithm will converge in polynomial time (this holds since, as we said, in each step we always go to a better neighbor). For example consider the non-weighted versions of many optimization problems, such as Graph Partitioning without weights on the edges of the graph. In the general

case, where an exponential range of solution costs exists, there is no a priori bound better than the exponential. In such a case we want to know if the local search problem can be solved polynomially or not.

Finally, note that the quality of a solution s is measured by the ratio of its cost c_s with the cost of the global optimum c_{s^*}, ($\lambda = \frac{c_s}{c_{s^*}}$). We say that a neighborhood structure guarantees ratio ε or that it is ε-approximate, if for every instance x and locally optimal solution $\hat{s}$, the cost of $\hat{s}$ is at most greater than a factor ε of the minimum cost ($\lambda \leq \varepsilon$). Because it must be $\varepsilon \geq 1$, at the maximization problems we have: $\lambda = \frac{c_{s^*}}{c_s}$ with $\lambda \leq \varepsilon$.

3 Some optimization problems and their usual neighborhoods

The following optimization problems are NP-complete and their corresponding local search problems with some common neighborhoods are shown to be PLS-complete. Here we will give the definitions of the problems and some of those neighborhoods.

3.1 Graph Partitioning

Given a graph $G = (V, E)$ with $2n$ nodes and weights $w_e \in \mathbb{Z}^+$ on every edge, find a partition of the set of nodes V in two sets A, B where $|A| = |B| = n$, such that the cost of the cut $w(A, B)$ is minimized. (The maximization version of graph partitioning is equivalent to the minimization version both as optimization and as local search problems under all the following neighborhoods, appropriately modified, [4])

A very simple neighborhood is *Swap* and, as we previously saw, is defined as follows: a partition (A, B) of the set of nodes V into two equal parts has as neighbors all the partitions, which can be produced by swapping a node in A with a node in B.

A much more complex neighborhood is *Kernighan-Lin*. The heuristic that explores this neighborhood moves from one partition to a neighboring one through a sequence of greedy swaps. At each step of the sequence we choose to swap the best pair of nodes among those that have not been moved in previous steps of the sequence. By the term "best" we imply that the swap produces the minimum cost differential, i.e. the weight of the cut decreases the most or increases the least. The opposite holds for maximization problems. In case of a tie, a tie-breaking rule is used to chose only one pair. Thus, a sequence of partitions $(A_i, B_i), i = 1, \ldots, n$ from a partition (A, B) is formed, where $|A_i - A| = |B_i - B| = i$. All these partitions are neighbors of the initial one.

Obviously this neighborhood is more powerful than the simple Swap. Observe that if a partition is a local optimum under Kernighan-Lin then it will be a local optimum under

Swap too, because in the opposite case its first neighbor would be better. But as we will see, both problems are polynomially equivalent.

Another neighborhood, usually used with this problem is *Fiduccia-Mattheyses (FM)*. FM is like Kernighan-Lin with the only difference that each step of the sequence consists of two substeps. In the first substep we move the "best" unmoved node from one side to the other and in the second substep we move the "best" unmoved node of the opposite side to have a balanced partition again. *FM-Swap* is one more neighborhood, obtained from the FM if we use only the first step of FM's sequence of steps.

3.2 Travelling Salesperson Problem

In TSP we are given a complete graph of n nodes ('cities') with positive integer weights ('distances') w_e on each edge and we want to find a least-weight tour that passes exactly once through each city or equivalently we are looking for a simple circle of length n with minimum weight. If the graph is not complete then it might not have a circle of length n. The *Hamiltonian cycle* problem is to find if a graph with n nodes has a simple circle of length n and it is NP-complete. The condition under which at least one of the Hamiltonian cycles is always a cycle of minimum weight, is the weights to satisfy the triangular inequality (i.e. $w_{ij} \leq w_{ik} + w_{kj}$). Then the problem is called *metric TSP*. A more specific case is when the cities are points on the plane and the weights of the edges are their Euclidean distances. If the weights of the edges are not symmetric then we have a directed graph and the problem is called *Asymmetric TSP*. All these problems are NP-complete.

Considering the initial TSP, we can define a whole set of neighborhoods called *k-Opt*, with $k \geq 2$ in general. The neighbors from these neighborhoods are obtained as follows. Starting from an arbitrary Hamiltonian cycle we delete k edges in order to obtain k non-connected paths. Then we reconnect these k paths such that a new tour is produced. The locally optimal solution is then called k-Opt solution. This neighborhoods can be used both on symmetric and asymmetric problems. As we will see later, TSP/k-Opt is PLS-complete for a fixed k, but we don't know anything about small values of k such as in the cases of TSP/2-Opt and TSP/3-Opt.

For the TSP problem there is a neighborhood that permits the replacement of an arbitrary number of edges between two neighboring tours using a greedy criterion to stop and is called *Lin-Kernighan*. The main idea is as follows: Given a tour we delete an edge (a, b) to obtain a Hamiltonian path with ends a and b. Let a be stable and b variable. If we add an edge (b, c) from the variable end, then a circle is created. There is a unique edge (c, d) that incidents on c, whose deletion breaks the circle, producing a new Hamiltonian path with a new variable end d. This procedure is called rotation. We can always close a tour by adding an edge between the stable end, a, and the variable end, d. Thus,

a movement of the Lin-Kernighan heuristic from a tour to a neighboring one consists of a deletion of an edge, a greedy number of rotations and then the connection of the two ends. There are a lot of variations of this main schema depending on how we choose a rotation. Such a variation is the *LK'* neighborhood under which the TSP has been shown to be PLS-complete, [5]. It is open if the TSP/Lin-Kernighan is PLS-complete or not.

3.3 Max-Cut

Given an undirected graph $G = (V, E)$ with positive weights on its edges, find a partition of the set of nodes V into two (not necessarily equal) sets, whose cut $w = (A, B)$ has the maximum cost. The version of minimization (Min-Cut) can be solved in polynomial time while the Max-Cut is NP-complete. The simplest neighborhood for Max-Cut is Flip, with which two solutions (partitions) are neighbors if the one can be obtained from the other by moving one node from the one side of the partition to the other. A Kernighan-Lin neighborhood can be defined for this problem, too, where the sequence of steps is a sequence of flips of nodes.

3.4 Max-Sat

In maximum satisfiability *(Max-Sat)* we have a boolean formula in conjunctive normal form (CNF) with a positive integer weight for each clause. A solution is an assignment of 0 or 1 to all the variables. Its cost, to be maximized, is the sum of the weights of the clauses that are satisfied by the assignment. *Max-k-Sat* is the same problem with the restriction of at most (or sometimes exactly) k literals in each clause. The simplest neighborhood for this problem is also the Flip neighborhood, where two solutions are neighbors if one can be obtained from the other by flipping the value of one variable. A Kernighan-Lin neighborhood can be defined for this problem, too, where the sequence of steps is a sequence of flips of variables.

3.5 Not-all-equal Max-Sat

An instance of not-all-equal maximum satisfiability (*NAE Max-Sat*) consists of clauses of the form NAE($\alpha_1, \ldots, \alpha_k$), where each α_i is either a literal or a constant (0 or 1). Such clauses are satisfied if their elements do not all have the same value. Each clause is assigned a positive integer weight. A solution is again an assignment of 0 or 1 to all the variables and its cost, to be maximized, is the sum of the weights of the satisfied clauses. If we restrict the clauses to have at most (or sometimes exactly) k literals then we have the *NAE Max-k-Sat* problem. The restriction to instances with no negative literals

in their clauses is called *Pos NAE Max-Sat*. We can define the Flip neighborhood and the Kernighan-Lin neighborhoods for this problem as for Max-Sat.

3.6 Stable configurations in neural networks of Hopfield type models

We are given a non-directed graph $G = (V, E)$ with a positive or negative weight w_e on each edge e and a threshold t_v for each node v (we can assume that the missing edges have weight equal to 0). A configuration assigns to each node v a state s_v, which is either 1 ('on') or -1 ('off'). These values can also be 1 and 0 but both versions are equivalent. A node is "happy" if $s_v = 1$ and $\sum_u w_{(u,v)} s_u s_v + t_v \geq 0$ or $s_v = -1$ and $\sum_u w_{(u,v)} s_u s_v + t_v \leq 0$. A configuration is stable if all the nodes are happy. The problem is to find a stable configuration for a given network. It is not obvious, a priori, that such a configuration exists. Actually, at the case of the directed graphs it is possible not to exist.

Hopfield, [6], showed that in the case of the undirected graphs always exists such a configuration. To prove this, he introduced a cost function $\sum_{(u,v)\in E} w_{(u,v)}\ s_u s_v + \sum_{v\in V} t_v s_v$ and argued that if a node is unhappy then changing its state will crease the cost. This means that the stable configuration coincide with the local optimum for this function under this "Flip" neighborhood, and a local optimum, of course, always exists.

If all the edge weights are negative, then the stable configuration problem is equivalent to $s-t$ Max-Cut/Flip, while if all the edges are positive the problem is equivalent to $s-t$ Min-Cut/Flip, (s and t are two nodes that must be on different sides of the partition). Additionally, if all the thresholds are 0 then the stable configuration problem is equivalent to Max-Cut/Flip or Min-Cut/Flip, depending on whether the edge weights are negative or positive, respectively, (see [7, 8, 9]). The $s-t$ Min-Cut and Min-Cut problems can be optimally solved in polynomial time, hence the stable configuration problem of neural nets with positive weights is solved polynomially. For the special case of Max-Cut/Flip for cubic graphs we can find local optima in polynomial time.

4 The PLS class

PLS class was designed to include those Local Search problems which are related to the usual local search heuristics. All these problems have the following properties in common. They find initial solutions, compute solution costs and search a neighborhood "easily", that is in polynomial time.

Typically PLS is defined as follows: Consider a local search problem Π. We assume that its instances are coded in binary strings and for every instance x, its solutions $s \in \mathcal{J}_\Pi(x)$ are binary strings, too, with their length bounded by a polynomial on x's length. Without loss of generality we can, also, assume that all the solutions are coded as strings

of the same length $p(|x|)$. Finally, we assume, for simplification, that the costs are non-negative integers (this theory can be straightforwardly expanded to rational costs, too).

Definition 4.1 *A Local Search problem Π is in PLS if there are three polynomial time algorithms A_Π, B_Π, C_Π with the following properties:*

1. *Given a string $x \in \{0,1\}^*$, algorithm A_Π defines if x is an instance of Π and in that case produces a solution $s_0 \in \mathcal{J}_\Pi(x)$.*
2. *Given an instance x and a string s, algorithm B_Π defines if $s \in \mathcal{J}_\Pi(x)$ and if it is so, then it computes the cost $f_\Pi(s,x)$ of the solution s.*
3. *Finally, given an instance x and a solution s, algorithm C_Π defines if s is a local optimum, and if it is not C_Π gives a neighbor $s' \in \mathcal{N}_\Pi(s,x)$ with a strictly better cost, i.e. $f_\Pi(s',x) < f_\Pi(s,x)$ for a minimization problem and $f_\Pi(s',x) > f_\Pi(s,x)$ for a maximization problem.*

All common local search problems, are in PLS. From the definition we can construct a local search algorithm which starts with the initial solution $s_0 = A_\Pi(x)$ and applies iteratively algorithm C_Π until it reaches a local optimum.

Having defined the PLS class, we are wondering where it lies in relation to the other known classes. The largest part of the Complexity Theory relies on the Decision Problems ($|O_\Pi(x)| = 1$, for each x - "YES-NO" Questions). In particular, the fundamental classes P and NP are classes of the Decision Problems. Usually these two classes are sufficient for the determination of the complexity of the Optimization Problems. On the one hand, we can prove that an Optimization Problem is "easy" if we can show a polynomial time algorithm that solves it. On the other hand if the optimization problem is "hard" we can usually show this by transforming it to a related decision problem, OP-decision, and show the latter to be NP-complete.

The OP-decision problem is defined as follows:

> Given an instance x and a cost c, is there any solution $s \in \mathcal{J}_{OP}(x)$ with cost at least as "good" as c? ($f_{OP}(s,x) \leq c$ for minimization problems, $f_{OP}(s,x) \geq c$ for maximization problems)

Obviously the OP-decision problem is not harder than the OP since an algorithm that solves the OP can be used to solve the OP-decision. If the OP-decision problem is NP-complete then the Optimization Problem will be NP-hard. This means that there is an algorithm for an NP-complete problem which uses an algorithm for the optimization problem as a subroutine and takes polynomial time, apart from the running time of the subroutine's calls.

In Local Search problems unfortunately it does not seem to be such a transformation which gives a proper Decision Problem not harder than the initial one. For that reason the classes P and NP of the decision problems cannot characterize the complexity of the local search problems and we should examine them from the beginning as simple Search Problems.

We define classes NP_S and P_S, which are the search analogues of classes NP and P, as follows:

- NP_S is the class of Search Problems (Relations) $\mathbb{R} \subseteq \{0,1\}^* \times \{0,1\}^*$ which are polynomially bounded and polynomially recognizable. Meaning that
  - if $(x, y) \in \mathbb{R}$ then $|y|$ is polynomially bounded in $|x|$ and
  - there is a polynomial time algorithm that given a pair (x, y) determines whether it belongs to $\mathbb{R}$ or not.
- Such a problem $\mathbb{R}$ is in P_S if there is a polynomial time algorithm that solves it, i.e., given an instance x, either produces as output a y such that $(x, y) \in \mathbb{R}$ or it reports (correctly) that there is no such y.

Easily follows from the definitions that

Proposition 4.1 *$P_S = NP_S$ if and only if $P = NP$*

PLS lies somewhere between P_S and NP_S. On the one hand, we see that any problem in P_S can be formulated as a PLS-problem. That is, we can define for each instance x of $\mathbb{R}$ a set of solutions $\mathcal{J}(x)$, a cost function $f(y, x)$ and a neighborhood function $\mathcal{N}(y, x)$ along with the corresponding algorithms A, B, C, which will satisfy the conditions in the definition of the PLS, such that $(x, y) \in \mathbb{R}$ if and only if y is a local optimum for x. Simply, let $\mathcal{J}(x) = y : (x, y) \in \mathbb{R}$ and for each $y \in \mathcal{J}(x)$ let $f(y, x) = 0$ and $\mathcal{N}(y, x) = y$. Algorithm A of definition 4.1 is the polynomial algorithm that solves the problem $\mathbb{R}$, algorithm B uses the algorithm that recognizes the members of $\mathbb{R}$ and algorithm C is trivial.

On the other hand, every problem Π in PLS is also in NP_S The relation $\{(x, y) : y$ is locally optimal for $x\}$ is polynomially recognizable from the algorithm C_Π of definition 4.1. Hence we have the following theorem

Theorem 4.1 $P_S \subseteq PLS \subseteq NP_S$

Observing the previous theorem we are now wondering if we can conclude in a more tight relation. The question now is if PLS coincides with any of its bounds in the above

relation or if it is properly between them. On the lower side it is not clear if PLS can be equal to P_S. Although we cannot conclude anything about this using the current computer theory, such a result would be remarkable, since it would require a general method for finding local optima at least as clever as the ellipsoid algorithm for the Linear Programming, which is one of the simplest and with very good behavior member of PLS.

On the upper side we have strong complexity theoretic evidence of proper containment. We know that NP_S includes NP-hard problems. For example the relation that consists of the pairs {x= a graph, y= a Hamilton cycle of x} is in NP_S. It is NP-hard to solve this search problem since it includes the solution of the Hamiltonian cycle problem, which is a decision NP-complete problem. The following fact shows that it is very unlikely that PLS contains NP-hard problems.

Theorem 4.2 *If a PLS problem Π is NP-hard then NP=co-NP.*

Proof. If Π is NP-hard then there is, by definition, an algorithm M for any NP-complete problem X, that calls an algorithm for Π as a subroutine and takes polynomial running time, apart from the time spent during the subroutine calls. But then we can verify that a given string x is a 'no'-instance of X in non-deterministic polynomial time as follows: just guess a computation of M with input x, including the inputs and the outputs of the calls to the subroutine for Π. The validity of the computation of M outside of the subroutines can be checked in deterministic polynomial time , by our assumption of M. The validity of the subroutine outputs can be verified using the polynomial time algorithm C_Π, whose existence is implied by the fact that Π is in PLS, to check whether the output is really a locally optimal solution for the input. Thus the set of 'no'-instances of X is in NP, i.e. $X \in co - NP$. Since X is NP-complete, it is implied that $NP = co - NP$. □

Formerly we saw that we cannot use NP-hardness to relate the Local Search problems to the class NP and argue that they are intractable, as we do with the Optimization problems. Therefore, in order to achieve something similar, we will relate them to each other with proper reductions and we will identify the hardest problems in PLS.

Definition 4.2 *Let Π_1 and Π_2 two Local Search problems. A PLS-reduction from Π_1 to Π_2 consists of two polynomial time computable functions h and g such that:*

1. *h maps instances x of Π_1 to instances $h(x)$ of Π_2*
2. *g maps pairs of the form (solution of $h(x)$,x) to solutions of x and*
3. *for all instances of Π_1, if s is local optimum for the instance $h(x)$ of Π_2 then $g(s, x)$ is a local optimum of x.*

If there is such a reduction, then we say that Π_1 PLS-reduces to Π_2.

Notice that, by its definition, the two requisitions that g satisfies are to map every solution t_i of $h(x)$ to one solution s_j of x, and if t_i is a local optimum of $h(x)$, then $g(t_i, x) = s_j$ is a local optimum of x. So we can argue that through a PLS-reduction we are transferred in an instance of a problem with, probably, fewer solutions and *local optima* than the initial one. The only case where $h(x)$ can have more solutions than x is when more than one of the solutions of the $h(x)$ are mapped to only one solution of x.

It is easy to see that for the PLS-reductions the transitional property holds and that they allow us to relate the difficulty of a problem with that of another one.

Proposition 4.2 *If Π_1, Π_2 and Π_3 are problems in PLS such that Π_1 PLS-reduces to Π_2 and Π_2 PLS-reduces to Π_3, then Π_1 PLS-reduces to Π_3.*

Proposition 4.3 *If Π_1 and Π_2 are problems in PLS such that Π_1 PLS-reduces to Π_2 and if there is a polynomial time algorithm for finding local optima for Π_2, then there is also a polynomial-time algorithm for finding local optima for Π_1.*

Definition 4.3 *A problem Π, which is in PLS, is PLS-complete if every problem in PLS can PLS-reduce to it.*

We will now give the definition of the first problem which is showed in [4] to be PLS-complete. This problem is called *Circuit/Flip*. An instance of this problem is a combinatorial Boolean circuit x (more precisely its encoding) which consists of AND, OR and NOT gates or any other complete Boolean basis. Let x has m inputs and n outputs. The set of the solutions $\mathcal{J}(x)$ consists of all the binary strings of length m, i.e. all the possible inputs. The neighborhood $\mathcal{N}(s, x)$ of a solution s consists of all the binary strings of length m whose Hamming distance from s equals to 1. Remember that two binary strings have Hamming distance equal to one if they are different exactly in one bit. The cost of a solution s is the output vector of the circuit for input s, which expresses a number written in binary. More typically it is $f(s, x) = \sum_{j=1}^{n}(2^{j-1}y_j)$, where y_j is the jth output of the circuit with input s, reading from right to left. The problem can be defined either as a maximization or as a minimization problem, since as we will see soon the two versions are equivalent. Intuitively the local search problem asks for an input such that its output cannot be improved lexicographically by flipping a single input bit.

It is easy to see that Circuit/Flip is in PLS. Let algorithm A returns the vector of length m with all its digits equal to 1. Let algorithm B checks that the given binary string has length m and then computes the circuit x with input s. Finally, let algorithm C that computes the circuit x for all the input vectors with Hamming distance from s equal to 1 (there are only m of them). Algorithm C returns a vector if it has better cost than s. Hence from the Definition 4.1 we have that Circuit/Flip $\in$ PLS.

The maximization and the minimization versions of the Circuit/Flip problem are equivalent both for the local search problem and for the respective optimization problem. In order to convert an instance of one form to one of the other, which will have the same global and local optima, we simply add another level of logic in the circuit, which will flip the value of all the output variables (changes 1's to 0's and vice versa). Indeed, this transformation is a PLS-reduction from the one version to the other. Sometimes we use the prefixes Max- and Min- to clarify which version of Circuit/Flip we are referring to.

Theorem 4.3 *Both the maximization version and the minimization version of the Circuit/Flip problem are PLS-complete.*

The proof is omitted, here (see [4] for the complete proof), however, we will give some hints about it. First, it is showed that any problem L in PLS can be PLS-reduced to an intermediate problem Q, which has the same instances with L but its solutions and neighborhood function are the same with the Circuit/Flip problem. Thus, Q can be straightforwardly PLS-reduced to Circuit/Flip by making a circuit which computes the cost function of Q, with the same inputs and outputs.

We mentioned that Q has the same neighborhood with Circuit/Flip, i.e. the Flip neighborhood. This can be done, since any neighborhood of L, as complex as it is, will simply perturb, in a polynomial way, the bits that consist the encoding of a solution to produce new ones. Hence, all the complexity of the Q problem is shifted in its cost function. Problem Q overcomes the weakness of its very simple neighborhood, because it has three basic characteristics:

1. It corresponds to every solution s of a problem L a solution $ss00$, and a number of intermediate solutions with appropriate costs, such that there is access, through simple flips of one bit at a time, from $ss00$ to the correspondent solution of any perturbation s' of s.

2. It preserves the relative ordering of the solutions of L in respect to their costs. Hence, when a solution s of L has bigger (smaller) cost of another solution s', then all the corresponding solutions of s in Q will have bigger (smaller) costs from the corresponding solutions of s'.

3. Algorithm's B of Q definition is based on the algorithms B and C of L. Thus, the transition of Q's heuristic, from a group of solutions to a neighboring one, will be executed if and only if the L's heuristic would do the corresponding transition.

So, the Q problem has the same local and global optima with L, and their heuristics follow similar paths. Therefore the only differences between an instance of L and an instance of Q are the following:

- The costs of the solutions of Q are bigger, by a constant factor, from those of the corresponding solutions of L.
- Two neighboring solutions in L are not directly connected in Q, but through a unique obligate path. This path consists of a number of intermediate solutions, where each of them differs in one bit from its previous solution.

It has also been proved that

Theorem 4.4 *The following problems are PLS-complete:*

1. *Graph Partitioning under the Kernighan-Lin neighborhood (for every tie-breaking rule), [1], and under the following neighborhoods: (a) Swap, (b) Fidducia-Mattheyses, (c) FM-Swap, [10].*
2. *Travelling Salesman Problem under the k-Opt neighborhood for some fixed k, [11], and under the LK' neighborhood, [5].*
3. *Max-Cut/Flip, [10].*
4. *Max-2Sat/Flip, [12].*
5. *Pos NAE Max-3Sat/Flip, [10].*
6. *Stable configurations for neural networks, [10].*

5 Complexity of the standard local search algorithm

In this section we will be concerned with the running time of local search algorithms (Running Time Problem) and we will see how the theory of PLS-completeness can be adapted to study this issue. At first, we will give some definitions.

Definition 5.1 *Let Π a local search problem and let x an instance of Π. The* neighborhood graph $NG_\Pi(x)$ *of the instance x is a directed graph with a node for each feasible solution of x and an arc $s \to t$ when $t \in \mathcal{N}_\Pi(s, x)$. The* transition graph $TG_\Pi(x)$ *is the subgraph which includes all those arcs for which the cost $f_\Pi(t, x)$ is strictly better than $f_\Pi(s, x)$ (greater if Π is a maximization problem or smaller if Π is a minimization problem). The* height of a node v *is the length of the shortest path in $TG_\Pi(x)$ from v to a* sink, *that is a node with no outgoing arcs. The* height of $TG_\Pi(x)$ *is the greatest of the heights of its nodes.*

Since the transition graph expresses the additional information of the cost difference between two neighboring solutions, we can imagine a third dimension for the cost of each solution. Hence, we would obtain the transition graph from the neighborhood graph if we only kept the downward arcs, for a minimization problem, or the upward arcs for a maximization problem.

We will be concerned mainly with the transition graph. Note that $TG_{\Pi}(x)$ is an acyclic graph. Also see that the cost induces a topological ordering of the nodes: the arcs head from worst to better nodes. Hence, the local optima are the sinks of the graph. $TG_{\Pi}(x)$ represents the possible legal moves for a local search algorithm on instance x. Beginning from some node (solution) v, the standard local search algorithm follows a path from node to node until it reaches a sink. The length of that path is the number of the iterations of the algorithm, which determines its running time. The precise path that is been followed (and hence the complexity) is determined by the pivoting rule that we have chosen. At each node which is not a sink the pivoting rule chooses which of the outgoing arcs will be followed. The height of a node v is the lower bound on the number of iterations which are needed by the standard local search algorithm even if it uses at each iteration the best pivoting rule. Note that this rule may not be computable in polynomial time on the size of the instance (in general the transition graph has an exponential number of nodes).

If a local search problem has instances whose transition graph has exponential height, the standard local search algorithm will need exponential time in the worst case, regardless of how it chooses better neighbors. This turns out to be the case with all the problems that have been shown to be PLS-complete. The notion of PLS-reduction that we have defined is not adequate to prove this, but it can be strengthened in an appropriate way.

Definition 5.2 *Let P and Q two local search problems and let (h, g) a PLS-reduction from P to Q. We say that the reduction is* tight *if for every instance x of P we can choose a subset R from the feasible solutions of the image instance $y = h(x)$ of Q so that the following properties are satisfied:*

1. *R includes all local optima of y*
2. *For every solution p of x we can construct in polynomial time a solution $q \in R$ of y such that $g(q, x) = p$*
3. *Suppose that the transition graph of y, $TG_Q(y)$, includes a directed path from $q \in R$ to $q' \in R$, such that all the internal nodes of the path are outside R and let $p = g(q, x)$ and $p' = g(q', x)$ the respective solutions of x. Then either it will hold $p = p'$ or $TG_P(x)$ will include an arc from p to p'.*

See that the tight PLS-reduction is a PLS-reduction, i.e. all the solutions of $h(x)$ have a corresponding solution in x through g and the local optima of $h(x)$ correspond in local

optima of x. In addition, there is a subset R in $h(x)$ though, which includes all the local optima of $h(x)$ and all the solutions of x have at least one corresponding solution in R. Hence, x has fewer or equal number of solutions as the subset R of $h(x)$. Notice, also, that since all local optima are in R, all paths of $h(x)$ will end up in there.

The third property tells us that if a path gets out of R, then the solution that reaches when it comes again inside R cannot have smaller distance (in arcs-steps) from the initial than the distance between their corresponding solutions in x. By this restriction we ensure that the solutions outside R are not helpful for the decrease of the complexity of the problem with the local search method. Therefore

Lemma 5.1 *Suppose that P and Q are problems in PLS and that h, g define a tight PLS-reduction from the problem P to the problem Q. If x is an instance of P and $y = h(x)$ is its image in Q, then the height of $TG_Q(y)$ is at least as large as the height of $TG_P(x)$. Hence, if the standard local search algorithm of P takes exponential time in the worst case, then so does the standard algorithm for Q.*

Proof. Let x be an instance of P, let $TG_P(x)$ be its transition graph and let p be a solution (node) whose height is equal to the height of $TG_P(x)$. Let $y = h(x)$ and let $q \in R$ be a solution of y such that $g(q, x) = p$. We claim that the height of q in $TG_Q(y)$ is at least as large as the height of p in $TG_P(x)$. To see this, consider a shortest path from q to a sink of $TG_Q(y)$ and let the nodes of R that appear on this path be $q, q_1, \ldots, q_k$. Let $p_1, \ldots, p_k$ be the images under g of these solutions, i.e., $p_i = g(q_i, x)$. From the definition of a tight reduction, we know that q_k is a local optimum of y, and thus p_k is a local optimum of x. Also, for each i, either $p_i = p_{i+1}$ or there is an arc in $TG_P(x)$ from p_i to p_{i+1}. Therefore, there is a path of length at most k from node p to a sink of $TG_P(x)$. □

It is easy to see that we can compose tight reductions. Tight reductions allow us to transfer lower bounds of the running time of the local search algorithm from one problem to another. All PLS-complete problems that we have referred to are complete under tight reductions.

To prove that in the worst case the running time of the standard local search algorithm for the tightly PLS-complete problems is exponential, it suffices to show that there is a problem in PLS which has such a property.

Lemma 5.2 *There is a local search problem in PLS whose standard local search algorithm takes exponential time.*

Proof. Consider the following artificial minimization problem. For every instance x of size n, the set of solutions consists of all n-bit integers $0, \ldots, 2^n - 1$. For each solution i, its cost is i and if $i > 0$ then it has one neighbor, $i - 1$. Hence, there is a unique local and

global optimum, namely 0, and the transition graph is a path from $2^n - 1$ to 0. The local search algorithm, starting at $2^n - 1$, will follow this path and will stop after an exponential number of iterations. □

Theorem 5.1 *The standard local search algorithm takes exponential time, in the worst case, for all the problems referred in Theorem 4.4*

Note that the exponential bounds hold for every pivoting rule, including randomized and non-polynomially computed rules.

Hence we have a general approach for proving bounds on the complexity of the local search heuristics. Outside that, however, there have been very few results, based mostly on ad hoc methods and for particular pivoting rules.

6 The quality of local optima

As we have said, the local search method is usually applied to tackle hard optimization problems. By imposing a neighborhood structure upon the solutions of a problem and by searching a local, only, optimum we achieve to decrease the complexity of the problem. The only restriction in neighborhoods is that they must be searchable efficiently, that is in polynomial time. Ideally we would like to have an exact neighborhood, one in which the global and the local optima coincide. Unfortunately, something like that is rare and as intuitively one would understand, this decrease of complexity has an impact on the quality of the optima that we can guarantee to find. Typically we can say the following:

At first, remember that a problem is called *strongly* NP-hard if it remains NP-hard even when the weights (costs) of its instances are polynomially bounded.

Theorem 6.1 *Let Π an optimization problem and $\mathcal{N}$ a neighborhood function such that the local search problem $\Pi/\mathcal{N}$ is in PLS.*

1. *If Π is strongly NP-hard (respectively NP-hard), then $\mathcal{N}$ cannot be exact unless P=NP (resp. NP=co-NP).*
2. *If the approximation of Π within a factor ε is strongly NP-hard (resp. NP-hard), then $\mathcal{N}$ cannot guarantee ratio ε unless P=NP (resp. NP=co-NP).*

Proof. Let Π be a strongly NP-hard problem and lets consider an instance with polynomially bounded weights. Then the standard local search algorithm will converge in polynomial time. If, furthermore, the neighborhood is exact, then the solution that will be computed, will be a global optimum.

In general suppose that Π is an NP-hard (possibly, not strongly) optimization problem. Let it be a minimization problem. Typically the following decision problem is NP-complete: Given an instance x and a value v, is there any solution with cost at most v? If $\mathcal{N}$ is an exact neighborhood, then we can solve its complementary decision problem in non-deterministic polynomial time as follows: Given x and v, guess a solution $\hat{s}$ and verify that $\hat{s}$ is locally (therefore globally) optimum and that its cost satisfies the relation $f(\hat{s}) > v$. □

The analogous statements about the approximation ratio follow by the same arguments. All the problems that we have seen until now (TSP, Graph Partitioning, Max-Cut, Max-Sat) are strongly NP-hard.

So, we can't find a global optimum if an optimization problem is NP-hard or ε-approximate it if this approximation is NP-hard, unless NP=co-NP. In the previous section we also saw that the standard local search heuristic takes exponential time to find a local optimum for many interesting problems under a lot of usual neighborhoods (the PLS-complete problems).

A combination of these questions and an even weaker goal would be to guarantee an ε-approximation for any local optimum in polynomial time with the standard local search heuristic. In [3] it is proved that even such a guarantee cannot hold at least for the Circuit/Flip, the Graph Partitioning/KL and any other PLS-complete problem, that is shown complete under a tight and weight-preserving PLS-reduction.

Recently, in [13], Orlin et al introduced the notion of the ε-local optimum to be a solution $\bar{S}$, where $cost(\bar{S}) \leq (1+\varepsilon)cost(S)$, for all $S \in N(\bar{S})$. They, also, presented a "fully polynomial ε-local approximation algorithm", which finds such solutions in $O(n^2\varepsilon^{-1}logn)$.

7 Approximation results

Empirically, it is well known that the local search heuristics seem to produce very good approximate solutions. For example, the heuristic algorithms for TSP ends up to solutions which are very close to the global optimum, in "typical" instances of the problem, on the plane. They are even better than other algorithms which have better approximation performance in the worst case (for instance, the 3/2-approximation algorithm which presented in [14])

During the last years, a lot of local search algorithms were proved to give an ε-approximate solution of the global optimal solution of subcases of many characteristic and popular problems, in a rather competitive running time. This Section presents some results of this nature.

7.1 Max-k-Sat

Max-Sat is the first example of an NP-complete problem. It cannot, also, be in PTAS, unless P=NP. However, randomized local search solves 2-SAT in polynomial time, [15]. There is a lot of research in solving the k-SAT problem with "weakly exponential" algorithms. Recently, in [16], there was presented a deterministic local search method running in time $(2 - \frac{2}{k+1})^n$ up to a polynomial factor. The Max-k-Sat problem is defined, here, as the Max-Sat problem in Section 3.4, without weights, and an additional requirement of *exactly* k literals per clause.

Another set of neighborhoods used with this problem are the d-neighborhoods, meaning that the neighboring assignments have different values on up to d variables. Thus, the 1-neighborhood, also called Flip, is the neighborhood where only one variable of an assignment can be flipped, in order to obtain a neighboring one.

In [17], the following theorem, for this problem under the Flip neighborhood, is proved

Theorem 7.1 *Let m and m_{loc} be the number of satisfied clauses at a global and a local optimum, respectively, of any instance of the unweighted MAX-k-SAT. Then we have $m_{loc} \geq \frac{k}{k+1}m$, and this bound is sharp.*

Proof. Without loss of generality, we can assume that in the local optimum each variable is assigned the value true. If it is not the case, by putting $x'_i = \bar{x}_i$ if $x_i \leftarrow$ false, and $x'_i = x_i$ if $x_i \leftarrow$ true in the local optimum, we obtain an equivalent instance for which the assumption holds.

Let δ_i the variation of the number of satisfied clauses when variable x_i is flipped. Since the assignment is a local optimum, flipping any variable decreases the number of satisfied clauses, i.e. $\delta_i \leq 0$, for $1 \leq i \leq n$.

Let cov_s the subset of clauses that have exactly s literals matched by the current assignment, and $cov_s(l)$ the number of clauses in cov_s that contain literal l.

We have $\delta_i = -cov_1(x_i) + cov_0(\bar{x}_i)$. Indeed, when x_i is flipped from true to false one looses the clauses that contain x_i as the single matched literal, i.e. $cov_1(x_i)$ and gains the clauses that have no matched literal and that contain $\bar{x}_i$, i.e. $cov_0(\bar{x}_i)$.

After summing over all variables, we obtain $\sum_{i=1}^{n} \delta_i \leq 0$, thus $\sum_{i=1}^{n} cov_0(\bar{x}_i) \leq \sum_{i=1}^{n} cov_1(x_i)$. By using the following equality $\sum_{i=1}^{n} cov_1(x_i) = |cov_1|$ and $\sum_{i=1}^{n} cov_0(\bar{x}_i) = k|cov_0|$ which can be easily verified, we obtain $k|cov_0| \leq |cov_1| \leq m_{loc}$. Therefore $m = m_{loc} + |cov_0| \leq (1 + \frac{1}{k})m_{loc} = \frac{k+1}{k}m_{loc}$. □

Thus, the local search algorithm with the flip neighborhood is a $\frac{k}{k+1}-$ approximation algorithm for the unweighted MAX-k-SAT. Notice that this algorithm is polynomial, since the problem is unweighted.

A variation of local search is the *non-oblivious* local search, in which the original objective function of the problem is changed in order to guide the algorithm to better local optima. In [18] we have the following theorem.

Theorem 7.2 *The performance ratio*[2] *for any oblivious local search algorithm with a d-neighborhood for MAX-2-SAT is 2/3 for any* $d = O(n)$. *Non-oblivious local search with the flip neighborhood achieves a performance ratio* $1 - \frac{1}{2^k}$ *for MAX-*k*-SAT.*

For Max-2-Sat, for example, the non-oblivious objective function is a weighted linear combination of the number of clauses with one and two matched literals. Namely, $f_{NOB} = \frac{3}{2}|cov_1| + 2|cov_2|$, instead of the oblivious objective function $f_{OB} = |cov_1| + |cov_2|$. It can be shown that the above theorem cannot be improved by using a different weighted linear combination of $|cov_1|$ and $|cov_2|$.

In [19], better approximation algorithms can be obtained by using at first a non-oblivious local search algorithm, and then an oblivious local search algorithm starting with the solution obtained by the first algorithm.

7.2 Max-Cut

Formally, for the *unweighted Max-Cut* problem, we have the following definition: Given a graph $G = (V, E)$, find a partition (V_1, V_2) of V into disjoint sets V_1 and V_2, which maximizes the cardinality of the cut, i.e., the number of the edges with one end point in V_1 and one end point in V_2.

This problem was one of the first problems shown to be NP-complete, but it can be solvable in polynomial time for planar graphs and a few other special cases. It has also been proved that no 0.941-approximation[3] algorithm can exist, unless P=NP, [20].

The best approximation result so far is given from the Goemans and Williamson's algorithm and it is 0.878-approximative. This algorithm reformulates an integer program as a semidefinite program and solves it using a variation of the interior point method for linear programming, [21, 22]. However, it becomes very slow for instances with $n \geq 500$, and because of its complex design it cannot be easily implemented on dedicated circuits.

Considering the unweighted Maximum-Cut under the Flip neighborhood, we have the following theorem (see [23]). Notice that since it is unweighted, the heuristic will converge in polynomial time.

[2] the approximation ratio reversed here

[3] Sometimes, it is used the same definition of the approximation ratio for the maximization problems as that for the minimization (not the inverse), thus ε is smaller than one.

Theorem 7.3 *Given an instance x (a graph $G = (V, E)$) of the Max-Cut problem without weights on its edges, let (V_1, V_2) a locally optimal partition under the Flip neighborhood and let $m_A(x)$ the cost of the locally optimal partition of x under Flip. Then*

$$\frac{m^*(x)}{m_A(x)} \leq 2,$$

where $m^*(x)$ is the cost of the optimal partition.

Proof. Let (V_{1k}, V_{2k}) be the neighbor of a solution (V_1, V_2), by flipping vertex v_k from the one subset of nodes to the other. Let m the number of the edges of the graph. Since $m^*(x) \leq m$ it suffices to show that $m_A(x) \geq \frac{m}{2}$.

We symbolize with m_1 and m_2 the number of the edges that join the nodes inside V_1 and V_2, respectively, of a local optimum. We have

$$m = m_1 + m_2 + m_A(x). \tag{1}$$

Given any node v_i, we define

$$m_{1i} = \{v | v \in V_1 \text{ and } (v, v_i) \in E\}$$

and

$$m_{2i} = \{v | v \in V_2 \text{ and } (v, v_i) \in E\}$$

Since (V_1, V_2) is a local optimum, then for each node v_k the solution that comes up from the (V_{1k}, V_{2k}) has as value at most $m_A(x)$. This means that for each node $v_i \in V_1$,

$$|m_{1i}| - |m_{2i}| \leq 0$$

and for each node $v_j \in V_2$,

$$|m_{2j}| - |m_{1j}| \leq 0.$$

Summing up all the vertices in V_1 and V_2 we obtain

$$\sum_{v_i \in V_1} (|m_{1i}| - |m_{2i}|) = 2m_1 - m_A(x) \leq 0$$

and

$$\sum_{v_j \in V_2} (|m_{2j}| - |m_{1j}|) = 2m_2 - m_A(x) \leq 0.$$

Therefore, $m_1 + m_2 - m_A(x) \leq 0$. From this inequality and from equation 1 it is implied that $m_A(x) \geq m/2$ and the theorem is proved. □

Another algorithm, which uses the main idea of Goemans and Williamson's algorithm in combination with the local search method is LORENA introduced in [24]. It does not have the disadvantages of the Goemans and Williamson's algorithm (time consuming and difficulty on circuit implementation), but is only proved to be 0.39-approximative. Experimental results, though, show a much better approximative performance.

7.3 Travelling Salesperson Problem

The Travelling Salesperson Problem, defined in section 3.2, is a (strongly) NP-hard and PLS-complete problem. It has been shown, in [25], that there are local optima arbitrarily worst than the global optimum. It has also been shown in [26] that finding any polynomial time ε-approximation algorithm for TSP is NP-hard.

When the triangle inequality is present there is a $3/2$-approximation algorithm (see [14]). However the local search heuristic is used in practice, because it is faster and gives good approximation solutions in most cases. We, also, have the following result.

Theorem 7.4 (Chandra 99, [27]) *A local search algorithm with 2-Opt neighborhood achieves a $4\sqrt{n}$ approximation ratio for the* metric TSP.

Proof. Let $T(V)$ be any tour which is locally optimal with respect to the 2-opt neighborhood. Let E_k, for $k \in \{1, \ldots, n\}$ the set of big edges of $T(V) : E_k = \{e \in T(V) | wt(e) > \frac{2C_{opt}}{\sqrt{k}}\}$, where C_{opt} is the cost of the global optimum tour. Then the first part of the proof is to show that $|E_k| < k$. Assuming this last result is true, then it means that the weight of the k-th largest edge in $T(V)$ is at most $\frac{2C_{opt}}{\sqrt{k}}$, therefore

$$\begin{aligned} C &= \sum_{k=1}^{n} weight(\text{k-th largest edge}) \\ &\leq 2C_{opt} \sum_{k=1}^{n} \frac{1}{\sqrt{k}} \\ &\leq 2C_{opt} \int_{x=0}^{n} \frac{1}{\sqrt{x}}\, dx \\ &= 4\sqrt{n} C_{opt}. \end{aligned}$$

The proof of $|E_k| < k$ is by contradiction. Here we give only an idea of the proof. Give an orientation of the tour T. Let $t_1, \ldots, t_r$, with $r = |E_k| \geq k$ be the tails of each arc from E_k in tour $T(V)$. Then it can be shown that there exists at least $\sqrt{k}$ tails which are at a distance at least $C_{opt}/\sqrt{k}$ from each other. Consider the travelling salesman instance restricted on this set V' of tails. Then the shortest tour on this set has a length $C_{opt}(V')$ greater than $\sqrt{k}\frac{C_{opt}}{\sqrt{k}} = C_{opt}$ contradicting the fact that since the distances satisfy the triangular inequality then for any subset $V' \subseteq V$ one has $C_{opt}(V') \leq C_{opt}(V)$. □

This bound is tight to within a factor of 16. The above theorem combined with the result, also proved in [27], that for random Euclidean instances in the unit square, the expected number of iterations required by 2-Opt is $O(n^{10} logn)$, provides a proof of the quality of local search on such instances. However, for infinitely many n, the k-Opt algorithm can have a performance ratio that is *at least* $\frac{1}{4} n^{\frac{1}{2k}}$.

There is another case, called *TSP(1,2)*, in which every edge can have weight either one or two.

Theorem 7.5 (Khanna 94, [18]) *A local search algorithm with the 2-Opt neighborhood achieves a $3/2$-approximation ratio for TSP(1,2).*

Proof. Let $C = v_{\pi_1}, v_{\pi_2}, \ldots, v_{\pi_n, v_{\pi_1}}$ be a local optimum solution with the 2-opt neighborhood. Let O be any optimal solution. To each unit cost edge e in O we associate a unit cost edge e' in C as follows. Let $e = (v_{\pi_i}, v_{\pi_j})$ with $i < j$. If $j = i + 1$ then $e' = e$. Otherwise e' is a unit cost edge among $e_1 = (v_{\pi_i}, v_{\pi_{i+1}})$ and $e_2 = (v_{\pi_j}, v_{\pi_{j+1}})$. Indeed, either e_1 or e_2 must be of unit cost. If it is not the case, then the tour C', obtained from C by removing edges e_1 and e_2 and adding edges e and $f = (v_{\pi_{i+1}}, v_{\pi_{j+1}})$, has a cost at least one less than C and therefore C would not be a local optimal solution with the 2-opt neighborhood.

Let U_O denotes the set of unit cost edges in O and U_C the set of unit cost edges in C obtained from U_O using the above mapping. Since an edge $e' = (v_{\pi_i}, v_{\pi_{i+1}})$ in U_C can only be the image of unit cost edges incident on v_{π_i} in O and since O is a tour, there are at most two edges in U_O which map to e'. Thus $|U_C| \geq |U_O|/2$ and we obtain $\frac{cost(C)}{cost(O)} \leq \frac{|U_O|/2+2(n-|U_O|/2)}{|U_O|+2(n-|U_O|)} \leq \frac{3}{2}$. □

The above bound is shown to be asymptotically tight in [18].

7.4 Other Graph Problems

Let $G = (V, E)$ be an unweighted graph. A set system (or hypergraph) (S, C) consists of a base set S and a collection C of subsets (or hyperedges) of S. A k-set system is a set system where each set in C is of size at most k. We can also assign weights to the sets of C, as well as to the edges of G, in order to obtain a weighted set system or a weighted graph, respectively.

Now we can consider the following collection of problems:

- **3-Dimensional Matching** Given sets W, X, Y and a set $M \subseteq W \times X \times Y$, find a minimum cardinality matching, i.e. a subset $M' \subseteq M$ such that no two elements of M' agree in any coordinate.

- k**-Set Packing** Given a k-set system (S, C), find a maximum cardinality collection of disjoint sets in C. In a weighted set system we are looking for a maximum cost collection of disjoint sets in C.

- **Maximum Independent Set** Given a graph G, find a maximum cardinality subset of mutually non-adjacent vertices, i.e. a subset $V' \subseteq V$ such that $v_i, v_j \in V'$ implies $(v_i, v_j) \notin E$. In its weighted case (w-MIS), there are weights assigned to the nodes and we want a maximum weight subset of non-adjacent vertices.

- **Vertex Cover** Given a graph G find a minimum cardinality subset $V' \subseteq V$ such that every edge has at least one endpoint in V'.

- k-**Set Cover** Given a k-set system (S, C) find a minimum cover of S, i.e. a subset $C' \subseteq C$ of minimum cardinality such that every element of S belongs to at least one member of C'.

- **Color Saving** (or Graph Coloring) Given a graph G find an assignment of minimum number of colors to the vertices such that adjacent vertices are of different colors. The objective function is to minimize the total number of vertices minus the total number of colors used.

All these problems are NP-hard and MAX SNP-hard, in general. k-Set Packing is a generalization of Maximum Matching from sets of size two (i.e. edges) to sets of size $1, 2, \ldots, k$, hence, for $k = 2$, it is polynomially solvable even for its weighted case. Also, an Independent Set in the intersection graph $H(S, C)$ corresponds to a k-Set Packing in (S, C). Recall that the *intersection graph* $H(S, C)$ of a hypergraph (S, C) has a vertex for each hyperedge with two hyperedges adjacent if and only if they intersect (as sets). Note that the intersection graph of a k-set system C contains no $k + 1$-*claw*, i.e. no $k + 1$-independent set in the neighborhood of any vertex.

The k-Set Cover problem can be solved in polynomial time by matching techniques, for $k = 2$. For the general case, there is a simple greedy algorithm who has performance ratio $\mathcal{H}_k = \sum_{i=1}^{k} \frac{1}{i}$.

The k-*independent set system* of a graph is the collection of all sets of up to k independent vertices in G. An optimal Set Cover of the independent set system of a graph corresponds to an optimal Color Saving, with objective function just the number of colors used, but the size of the instance might have an exponential blowup.

The usual local search method for the unweighted cases of the above problems consists of t-improvements. That is, at each step, s new items are added in a solution, of a maximization problem, and at most $s - 1$ items are removed from it, for some $s \leq t$. For minimization problems holds the reverse.

The results that follow are obtained with some variation of the above local search neighborhood and are tight. In [28], there are similar results for more graph problems. So we have the following approximation ratios:

- $k/2 + \varepsilon$ for Maximum Independent Set problem in $k+1$-claw free graphs when $k \geq 4$, in time $O(n^{log_k 1/\varepsilon})$, and 5/3 for $k = 3$. This also applies to k-Set Packing and k-Dimensional Matching, [28].

- $(\Delta + 2 + 1/3)/4 + \varepsilon$ for Maximum Independent Set in graphs of maximum degree Δ, in time $\Delta^{O(\Delta log 1/\varepsilon)} n$, [28]. For $\Delta \geq 10$ there is also proved in [18], that an algorithm which outputs the larger solution of those computed by a local search and a greedy algorithm has performance ratio $(\sqrt{8\Delta^2 + 4\Delta + 1} - 2\Delta + 1)/2$.

- $\frac{2}{3}(d+1)$ for the w-MIS on d-claw free graphs, [29, 30], and $d/2$ with non-oblivious local search, [31].

- $2 - 2f_s(k)$ for Vertex Cover in $k+1$-claw free graphs, for $k \geq 6$, where $f_s(k) = (klnk - k + 1)/(k-1)^2 = 2 - (logk)/k(1 + o(1))$. For $k = 4, 5$ a ratio of 1.5 holds, [28].

- 4/3 for 3-Set Cover. For k-Set Cover, using a half greedy half local search algorithm, a ratio $\mathcal{H}_5 - 5/12$ holds, [32].

- 1.4 for Color Saving, [28]. For graphs with maximum independent sets of size 3 the performance ratio is 6/5, [32].

Another problem, well approximated with local search, is the minimum VFES problem, which is described below.

7.4.1 Minimum Vertex Feedback Edge Set

The graph-theoretic problem minimum VFES (Vertex Feedback Edge Set) is NP-hard and MAX SNP-hard. However, It is very useful in placing pressure meters in fluid networks or in any other system, formulated as a network, in which Kirchoff's laws are valid and a bijective relation exists between the flow and effort variables, like circuits and electrical networks.

We define the minimum VFES problem as follows: Given a graph, find a feedback edge set incident upon the minimum number of vertices. A feedback edge set is a subset of edges in a graph, whose deletion from the graph make the graph acyclic.

In [33], a $2 + \varepsilon$ approximation ratio is obtained, in $O(n^{O(1/\varepsilon)})$, by a local search algorithm is introduced, which can be made very efficient by restricting the neighborhoods to be searched, that is in $O(n^3 + n^2 f(\frac{1}{\varepsilon}))$, where f is an exponential function, but has no dependence on n. There is also presented a PTAS for the case of planar graphs. The neighborhood used by the algorithm is called k-Local Improvement. The current Feedback Edge Set (FES) has as neighbors all the FESs obtained by a replacement of at most

$k-1$ edges from the FES. The cost of a FES is the number of vertices incident to its edges.

7.5 Classification Problems with pairwise relationships

Generally, a classification problem consists of a set P of *objects* to be classified and a set L of *labels* (the classes). The goal is to assign a label to each object in a way that is consistent with some "observed data" that we have about the problem. Here we are interested about problems whose "observed data" are some pairwise relationships among the objects to be classified. These problems have been studied a lot since they are very useful in areas such as statistics, image processing, biometry, language modelling and categorization of hypertext documents.

A characteristic example, from image processing, is the *image restoration problem*. Consider a large grid of pixels. Each pixel has an "observed" intensity and a "true" intensity that we are trying to determine, since it was corrupted by noise. We would like to find the best way to label each pixel with a (true) intensity value, based on the observed intensities. Our determination of the "best" intensity is based on the trade-off between two competing influences: We would like to give each pixel an intensity close to what we have observed and - since real images are mainly smooth, with occasional boundary regions of sharp discontinuity - we would like spatially neighboring pixels to receive similar intensity values.

To be more precise we give the *Metric Labeling Problem* as defined by Kleinberg and Tardos in [34]. Consider a set P of n objects that we wish to classify and a set L of k possible labels. A *labeling* of P over L is simply a function $f : P \to L$. We choose a label for each object. The quality of our labeling is based on the contribution of two sets of terms

- For each object $p \in P$ and label $i \in L$, we have a non-negative *assignment* cost $c(p, i)$ associated with assigning the label i to the object p.
- We have a graph G over the vertex set P, with edge set E indicating the pairwise relationships among the objects. Each edge $e = \{p, q\}$ has a non-negative weight w_e, indicating the strength of this relation.

Moreover, we impose a distance $d(\cdot, \cdot)$ on the set L of labels. So if we assign label i to object p and label j to object q and $e = \{p, q\}$ is an edge of G, then we pay a *separation* cost $w_e d(i, j)$. Thus, the *total cost* of a discrete labeling f is given by

$$Q(F) = \sum_{p \in P} c(p, f(p)) + \sum_{e=\{p,q\} \in E} w_e d(f(p), f(q)).$$

The *labeling problem* asks for a discrete labeling of minimum total cost. Recall that a distance $d : L \times L \rightarrow \mathbb{R}^+$ is a symmetric function and $d(i, i) = 0$ for all $i \in L$. So, if d also satisfies the triangle inequality then d is a *metric*. Hence, the labeling problem is called metric labeling problem if the distance function $d(\cdot, \cdot)$ is a metric on the label set L. Two special cases of the metric d are the *uniform* metric, where $d(i, j) = 1$, for all $i \neq j$, and the *linear* metric, where $d(i, j) = |i - j|, i, j \in \mathbb{N}$. In fact, we can assume, without loss of generality, that the labels of the linear metric are integers $1, 2, \ldots, k$, since in the opposite case we can add the "missing" intermediate integers to the label set and set the cost of assigning them to any vertex to be infinite.

Considering the image restoration problem, with the pixels and their intensities, we can say that

- the assignment cost is getting bigger as the labels we examine become more different than the observed one, for a specific object p,
- the nodes of the graph G are the pixels and there are edges only between neighboring pixels, all with weight equal to 1,
- the distance function d indicates less similarity between two labels and is used to penalize different colors to adjacent pixels. We could simply choose the linear metric to distinguish labels in a grey-scale image, since the color values are integers, but this would lead to over-smoothness of the image and the object boundaries may become fuzzy. So, we would like a non-uniform robust metric, which will sufficiently penalize small differences in the color of neighboring pixels but after a value M, for which we are sure that it is an object boundary, the metric should give the same penalty. Hence, we use the *truncated linear metric* defined as $d(i, j) = min\{M, |i - j|\}$.

In [34], the metric labeling problem is related with other known and well studied problems. So, this problem can be viewed as an extension of the multi-way cut problem, in which we are given a weighted graph with k terminals and we must find a partition of the graph into k sets so that each terminal is in a separate set and the total weight of the edges cut is as small as possible. In the latter problem, there are the terminals which must receive a certain label while all the others do not care of what label they will get, so it is a special case of the metric labeling problem.

It can also be viewed as the *uncapacitated quadratic assignment problem*. In the quadratic assignment problem one must find a matching between a set of n given activities to n locations in a metric space so as to minimize a sum of assignment costs for activities and flow costs for activities that "interact". The metric labeling problem can be obtained from the quadratic assignment by dropping the requirement that at most one activity can

be sited at a given location. The activities then correspond to objects and the locations to labels, in the metric labeling problem.

Finally, there is a relation with a general class of Markov random fields. For a given set of objects P and labels L, the *random field* assigns for every labeling f, a probability PR[f]. The random field is *Markovian* if the conditional probability of the label assignment at object p depends only on the label assignments at the neighbors of p in G. If, additionally, the Markov Random Field satisfies the properties of *pairwise interactions* and "metric" *homogeneity*, then is called *metric Markov random field* (see [34] for more details). It is proved that the optimum of a metric labeling problem is equivalent to the optimal configuration of metric Markov random fields, but the transformation is not approximation preserving.

Although the metric labeling problem is NP-hard and MAX SNP-hard, there cases that can be solved polynomially. The cases of $l = 2$ labels (see [35, 36]) and that of the linear metric (see [37, 38, 39]) can be polynomially solved as two-terminal minimum cut problems. Also, Karzanof in [40, 41] showed some other special cases of the labeling problem to be polynomial. Boykof et al. [37] developed a direct reduction from labelings with uniform labelings to multiway cuts, but the reduction is not approximation preserving.

The approximability results obtained for the metric labeling problem and its subcases are the following:

- $O(log|L|loglog|L|)$, for general metrics, [34, 42]
- 2, for the uniform metric, [34, 42]
- 1, for the linear metric and distances on the line defined by convex functions (not necessarily metrics), [42]
- $2 + \sqrt{2} \simeq 3.414$ for the truncated linear metric, [42].

In [42], there is a $O(\sqrt{M})$-approximation result for the *truncated quadratic* distance function ($d(i, j) = min\{M, |i - j|^2\}$), also used in image restoration applications, which is not a metric function. There is also another result which essentially allows us to eliminate the label assignment cost function. That is, there is a reduction from the case with arbitrary assignment costs $c(p, i)$ to the case where $c(p, i) \in \{0, \infty\}$ for all p and i. The reduction preserves the graph G and the optimal solution, but increases the size of the label space from k to nk labels.

All the previous results are obtained by solving the relaxed version of an integer linear program, then rounding its solutions and measuring the gap between them and the solutions of the integer linear program. However, the linear programs involved are quite

large and this causes lots of these methods to be too slow and thus less practical. Here, we will present a local search method, which was showed by Gupta and Tardos in [43] to be *4-approximative of the truncated linear metric*, in polynomial time.

7.5.1 A 4-approximation local search method for the metric labeling problem with the truncated linear metric.

Recall that the truncated linear metric is defined as $d(i,j) = min\{M, |i-j|\}$, where i and j are from the set of labels $L = 1, 2, \ldots, l$. In a single local step we consider an interval I of labels of length at most M, and allow any subset of vertices to be related by any of the labels in I. Given a labeling f, we call another labeling f' a *local relabeling* if it can be obtained from f by a local move, i.e., if for all objects $f(p) \neq f'(p)$ implies that $f'(p) \in I$. Unfortunately, we are not able to find the best possible such local move because, as you can see, the neighborhood of a labeling is exponential. However it will be showed later that if the current labeling has cost sufficiently far above the minimum possible cost, then this method will find a move that significantly decreases the cost of the labeling.

In the algorithm we repeatedly pick a random interval I and try to relabel some subset of objects with labels from I, in order to decrease the cost of our labeling. After this local step, each object will either have its label unchanged or will have a label in the interval I. To perform this relabeling efficiently we will create a flow network and find a minimum s-t cut in it. This minimum cut can be associated with a new labeling f' and if f' has a lower cost than the cost of f, we move to the new labeling. In summary the algorithm is the following:

Algorithm Local Search
repeat
pick a random interval I
build the flow network N_I associated with I
if labeling given by the minimum cut
on N_I has lower cost
then move to new labeling
until a local optimum is reached.

The random intervals will be picked in the following manner: we pick a random integer $-M < r < l$, and set I to be the part of the interval of length M starting from offset r that lies in L, i.e. $I = \{r+1, r+2, \ldots, r+M\} \cap \{1, 2, \ldots, l\}$. Thus, we have a partition S_r of the label set with at most $\lceil l/M \rceil + 1$ intervals I in it, each of them with size exactly M, except from the initial and the final portion of the line, whose lengths might be smaller than M. Note that the probability, for any pair of labels $i, j \in L$, to lie in different intervals of the partition S_r is exactly $d(i,j)/M$. This algorithm can be

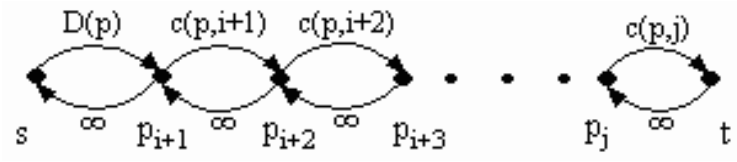

Figure 1: The chain for vertex p

trivially derandomized at a cost of a factor $(l + M)$ increase in the running time. This can be done by considering all possible $(l + M)$ intervals and, for instance, making the moves corresponding to the best possible interval.

The description of the flow network associated with an interval I to which the labels can be changed, follows. Let us consider that the labels in I are $\{i + 1, i + 2, \ldots, j\}$, with $(j - i) \leq M$. The flow network $N_I = (V, A)$ associated with I is a directed graph with a source s and a sink t, and with capacities on the edges. The first step, to construct it, is for each vertex p of the original graph G to add $(j - i)$ nodes, namely $\{p_{i+1}, \ldots, p_j\}$ to N_I (see Figure 1). We add directed edges (p_k, p_{k+1}) with capacity equal to the assignment cost $c(p, k)$, and directed edges (p_{k+1}, p_k) with infinite capacity, for $i + 1 \leq k \leq j$, where $p_{j+1} = t$. Finally, the edge (p_{i+1}, s) is assigned an infinite capacity, while (s, p_{i+1}) is assigned a capacity $D(p)$ which is defined as follows: if $f(p) \in I$ then $D(p) = \infty$ else $D(p) = c(p, f(p))$.

This construction captures the assignment cost. To see this, consider any minimum s-t cut in N_I. The infinite capacity edges ensure that this cut will include exactly one edge from the chain corresponding to each vertex p. If edge (p_k, p_{k+1}) is cut this means that the vertex p is assigned the label k, unless (p_s, p_{i+1}) is cut and $f(p) \notin I$, where the original label is retained for vertex p. Hence, the assignment cost is exactly the capacity of the edge in the cut.

The second step of the construction is to model the separation cost. Let $e = \{p, q\}$ be an edge of the original graph. Depending on the labels of the vertices p, q we have the following cases: 1) If both $f(p)$ and $f(q)$ are not in I, then for each of the corresponding nodes p_k and $q_k, i + 1 < k \leq j$, we add a pair of oppositely directed edges between them, each with capacity w_e. We also add a new node υ_{pq} and connect it with the nodes p_{i+1}, q_{i+1} with oppositely directed edges with capacities $w_e d(f(p), i + 1)$ and $w_e d(f(q), i + 1)$, respectively. Finally, we add an edge (s, υ_{pq}) with capacity $w_e d(f(p), f(q))$, (see Figure 2). 2) If both $f(p)$ and $f(q)$ are in I, then we do nothing. 3) If $f(p) \notin I$ but $f(q) \in I$, then we add an edge (p_{i+1}, q_{i+1}) with capacity $w_e d(f(p), i + 1)$.

This structure captures the separation costs. Let f' be the labeling corresponding to the cut, and let us focus on the edge $e = \{p, q\}$. If both vertices retain their original labels, then the cut will be minimized when it passes through (s, υ_{pq}), and incurs a cost of

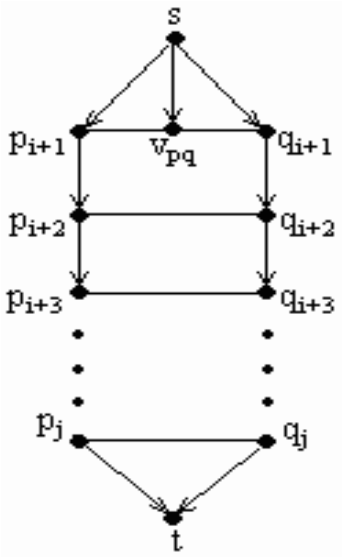

Figure 2: Construction for the edge $\{p, q\}$

$w_e d(f(p), f(q))$. If both vertices are labeled with labels in I, then the cut will be exactly $w_e|f'(p) - f'(q)|$. For the above cases, the cuts equal exactly the separation costs. If one of the vertices (say p) retains its label $f(p) \notin I$ and q is labeled with a new label in I, $f'(q) = k \in I$, then the cut will incur a separation cost $w_e[d(f(p), i+1) + (k - (i+1))]$, which possibly overestimates the actual separation cost in the new labeling.

Note that a minimum cut in N_I can contain at most one of the edges connected with v_{pq}, since only one edge in any of the two pairs of opposite edges can be in the cut, and that in any set of up to three permissible edges, the cost of any two edges is more than the cost of the third one, because $d(\cdot, \cdot)$ satisfies the triangular inequality. A cut is called *simple* if it has finite capacity and it does not cut more than one of the above five edges associated with any edge $e \in E$.

Theorem 7.6 *The simple cuts in the flow network N_I are in one-to-one correspondence with local relabelings f'. The cost of the relabeling $Q(f')$ is no more than the cost of the associated cut, and the cost of the cut overestimates the cost of the labeling by replacing the separation cost $w_e d(f'(p), f(p))$ for edges $e = \{p, q\}$ where exactly one end receives a label in I by a possibly larger term $w_e[d(f'(p), i+1) + d(i+1, f'(q))]$. Further, we have that*

$$d(f'(p), f'(q)) \leq d(f'(p), i+1) + d(i+1, f'(q)) \leq 2M. \quad (2)$$

We will now sketch the proof of the 4-approximability of this method. First we give some definitions. Let f^* be a fixed optimal labeling and let the algorithm's current label-

ing be f. For any subset $X \subseteq P$, let $A^*(X)$ and $A(X)$ be the assignment cost that the optimum and the current labeling pay respectively for the vertices in X, and for a set of edges $Y \subseteq E$, let $S^*(Y)$ and $S(Y)$ be the separation cost for those edges paid by the optimum and the current solution respectively. So, $Q(f) = A(P) + S(E)$ and the optimum $Q(f^*) = A^*(P) + S^*(E)$.

Consider the case when the algorithm chooses an interval I. Let P_I be the set of vertices of G to which f^* assigns labels from the interval I. Let E_I be the set of edges in $E(G)$ such that the f^*-labels of both endpoints lie in I. Let ϑ_I^- be the set of edges such that exactly one end of the edge has f^*-label in I, the end with higher f^*-label, and ϑ_I^+ be the set of edges that only the end with lower f^*-label has an f^*-label in I. In the proof it is considered a random partition S_r of the labels. Clearly, $P = \cup_{I \in S_r} P_I$ and $\cup_{I \in S_r} \vartheta_I^- = \cup_{I \in S_r} \vartheta_I^+$. ϑ_r is used to denote this union of the *boundary edges* in the partition. Also note that $E = \vartheta_r \cup (\cup_{I \in S_r} E_I)$.

The following lemmas are used in the proof of theorem 7.7. For their proofs see [43].

Lemma 7.1 *For a random partition S_r, the expected value of $M \sum_{e \in \vartheta_r} w_e$ is $S^*(E)$.*

Lemma 7.2 *For a labeling f, and an interval I, the local relabeling move that corresponds to the minimum cut in N_I decreases the cost of the solution by at least*

$$(A(P_I) + S(E_I \cup \vartheta_I^- \cup \vartheta_I^+)) - (A^*(P_I) + S^*(E_I \cup \vartheta_I^-) + M \sum_{e \in \vartheta_I^-} w_e + 2M \sum_{e \in \vartheta_I^+} w_e).$$

Theorem 7.7 *If the labeling f is a local optimum, its cost $Q(f)$ is at most 4 times the optimal cost $Q(f^*)$.*

Proof. The fact that f is a local optimum implies that the improvement indicated by the lemma 7.2 is non-positive for any interval I, i.e., for all I

$$A(P_I) + S(E_I \cup \vartheta_I^- \cup \vartheta_I^+) \leq A^*(P_I) + S^*(E_I \cup \vartheta_I^-) + M \sum_{e \in \vartheta_I^-} w_e + 2M \sum_{e \in \vartheta_I^+} w_e. \quad (3)$$

Now consider a partition S_r and sum these inequalities for each interval $I \in S_r$. On the left hand side we get $A(P) + S(E) + S(\vartheta_r)$ as edges in ϑ_r occur in the boundary of two intervals. This is at least $Q(f)$, the cost of the labeling f. Summing the right hand side, we get exactly $A^*(P) + S^*(E) + 3M \sum_{e \in \vartheta_r} w_e$. So we have that

$$Q(f) \leq A^*(P) + S^*(E) + 3M \sum_{e \in \vartheta_r} w_e$$

for any partition S_r. Taking expectations, the left side is a constant and by lemma 7.1, the expected value of the right hand side is at most $A^*(P) + 4S^*(E)$. Thus we get $Q(f) \leq A^*(P) + 4S^*(E) \leq 4Q(f^*)$. □

7.6 k-Median and Facility Location Problems

There are a lot of different versions of the k-median and facility location problems. To give a general framework for these problems we have to follow a top-down procedure, adding each time the specific requirements that each variation has. There is also the k-means problem, which is very similar to the k-median problem, and it is defined at the end of this section.

Generally, let $N = \{1, 2, \ldots, n\}$ be a subset of *locations* and $F \subseteq N$ be a set of locations at which we may *open* a *facility*. Each location $j \in N$ has a demand d_j that must be shipped to j. For any two locations i and j, let c_{ij} denote *the cost of shipping a unit of demand* from i to j. In these problems the goal is to identify a set of open facilities $S \subseteq F$ and an assignment of locations to S, such that some *cost function* is minimized. The cases where all the unit shipping costs are assumed to be nonnegative, symmetric and satisfy the triangle inequality, are the *metric versions* of the problems, for which all the following results are obtained.

In the *facility location problem* (UFL) we are, also, given a non-negative cost f_i of opening a facility at i, for every location $i \in F$. The cost function that has to be minimized for this problem is the sum of the cost of opening the facilities (*facility cost*) and the *shipping (or service) cost*. On the other hand, in the *k-median problem*, instead of facility costs we are just restricted to minimize the service cost, opening at most k facilities ($|S| \leq k$).

More formally, the service cost associated with a set S of open facilities and an assignment of locations to them, $\sigma : N \to S$, is given by $C_s(S) = \sum_{j \in N} d_j c_{j\sigma(j)}$. The facility cost, for the UFL, is $C_f(S) = \sum_{i \in S} f_i$. For both problems, given a set S of open facilities, an assignment that minimizes the total cost is to assign each location $j \in N$ to the closest open facility in S. Thus, a solution to these problems is completely characterized by the set of open facilities S.

The previous problems are called *uncapacitated* (that is the "U" in "UFL") in the sense that the demand that can be shipped from any facility is infinite. The *capacitated* variants of the above two problems are divided in two categories depending on how the locations' demand can be served. If the demand of each location can be split across more than one facility then we have a *splittable capacitated* variant. If the demand of each location has to be shipped from a single facility then we have an *unsplittable capacitated* variant. Furthermore, the capacities that the facilities have can be *uniform* or *non-uniform*, that is either there is a common bound M for the capacities of all the facilities or each facility $j \in F$ has a specific capacity $u_j > 0$, respectively. For the case of capacitated problems with splittable demands, the assignment function changes and is given by $\sigma : N \times S \to \mathbb{R}$, where $\sigma(i, j)$ denotes the amount of demand shipped to location i from facility j.

As we have already mentioned, in the uncapacitated problems, given a set of open fa-

cilities, an optimal assignment is obtained by simply assigning each location to its closest open facility. In the capacitated variations such an assignment may violate the capacity constraint(s). Fortunately, for the splittable capacitated problems we can compute an optimal assignment in polynomial time, solving an appropriately defined instance of the transportation problem, [44]. However, when the demands are unsplittable, it is NP-hard to compute an optimal assignment for a given set S of open facilities. Therefore, we require a solution to one of the capacitated problems with unsplittable demands to specify a feasible assignment together with the set of open facilities.

Other variations of the capacitated problems are those, which permit multiple *copies* of a facility to be opened in a location. Hence, in that case we are looking for a multi-set S of open facilities. The difference with the uncapacitated problems is that we are only permitted to open at most m copies. Thus, the capacitated facility location problem with at most m copies of each facility permitted, is called *m-CFLP*. The notion of copies is equivalent with that of *capacity blowup*, considered in [45], which is used in capacitated k-median problems.

To sum up, the problems that we have defined are the metric versions of the following problems:

1. uncapacitated k-median and facility location problems (UFL),
2. capacitated k-median and facility location problems with unsplittable demands having uniform or non-uniform capacities, with copies (m-CFLP) or not,
3. capacitated k-median and facility location problems with splittable demands having uniform or non-uniform capacities, with copies (m-CFLP) or not.

In Table 1 we give some approximation ratios for some of these metric (except from the third one which is general) problems, a bound of copies (or capacity blowup) if they are permitted and the reference. Note that for the k-median problems an (a, b)-*approximation algorithm* is defined as a polynomial time algorithm that computes a solution using at most bk facilities and with cost at most a times the cost of an optimal solution using at most k facilities.

We should also refer a theorem proved in [53]. It says that there is a polynomial algorithm which, given a solution S to the k-CFLP, produces a solution $\hat{S}$ to the 2-CFLP at additional cost at most twice the optimal value of a solution to the 1-CFLP.

The results, on Table 1, with an asterisk next to their reference, are obtained with local search and some of them are the best known. Local search seems to work very well with these problems. We will give, now, such an algorithm with its ε-approximability proof. It is the $5(1+\varepsilon)$-approximation algorithm for the uncapacitated k-median problem, presented in [46], whose extension gives a $(3 + 2/p)(1 + \varepsilon)$-approximation algorithm.

Problem	Bound	Copies (Capacity Blowup)	Reference
1. uncapacitated k-median	$(1+\varepsilon, 3+5/\varepsilon)$	-	[45]*
	$(1+5/\varepsilon, 3+\varepsilon)$	-	[45]*
	$5(1+\varepsilon)$	-	[46]*
	$(3+2/p)(1+\varepsilon)$	-	[46]*
	$O(logkloglogk)$	-	[47]
	$(1+\varepsilon, (1+1/\varepsilon)(lnn+1))$	-	[48]
	$(2(1+\varepsilon), 1+1/\varepsilon)$	-	[49]
2. Euclidean k-median	$(1+\varepsilon, 1)$	-	[50]
3. general uncapacitated facility location	$O(logn)$	-	[51]
4. uncapacitated facility location	$3(1+\varepsilon)$	-	[46]*
	1.74	-	[52]
	$\nexists 1.46$, unless $P = NP$	-	[53]
5. k-median, splittable, uniform	$(1+\varepsilon, 5+5/\varepsilon)$	none	[45]*
	$(1+5/\varepsilon, 5+\varepsilon)$	none	[45]*
6. k-median, unsplittable, uniform	$(1+\varepsilon, 5+5/\varepsilon)$	2	[45]*
	$(1+5/\varepsilon, 5+\varepsilon)$	2	[45]*
7. facility location, splittable, uniform	$8+\varepsilon$	none	[45]*
	$6(1+\varepsilon)$	none	[53]*
	7	7/2	[44]
	3	∞	[54]
	5	2	[53]
8. facility location, splittable, non-uniform	$4(1+\varepsilon)$	∞	[46]*
	$9+\varepsilon$	none	[55]*
9. facility location, unsplittable, uniform	$16+\varepsilon$	2	[45]*
	9	4	[44]

Table 1: Approximation bounds for k-median and facility location problems (references with asterisk indicate that the local search method was used). For the uncapacitated problems the notion of copies has no meaning so we put a '-'.

The notation in [46] is different, but equivalent with the one presented here, so we will redefine the problem.

7.6.1 Uncapacitated k-Median Problem

In the metric uncapacitated k-median problem, we are given two sets, F (facilities) and C (clients), and an input parameter $k, 0 < k \leq |F|$. There is a specified *metric distance* $c_{ij} \geq 0$ between every pair $i, j \in F \cup C$, which is used as *service cost*, too. The problem is to identify a subset $S \subseteq F$ of at most k facilities and to serve the clients in C by the facilities in S such that the total service cost is minimized. Thus, if a client $j \in C$ is served by (its closest) facility $\sigma(j) \in S$, then we want to minimize $cost(S) = \sum_{j \in C} c_{\sigma(j)j}$.

The general local search algorithm used in [46] is the following:

1. $S \leftarrow$ an arbitrary feasible solution.
2. While $\exists$ an operation op such that,
$$cost(op(S)) \leq (1 - \tfrac{\epsilon}{p(n,m)})cost(S),$$
do $S \leftarrow op(S)$.
3. return S.

where $n = |F|, m = |C|$ and $p(n, m)$ a polynomial in n and m. The neighborhood used in this local search procedure is *swap*. A swap is effected by closing a facility $s \in S$ and opening a facility $s' \notin S$. So

$$op(S) := S - s + s', \text{ for } s \in S \text{ and } s' \notin S.$$

and this swap will be denoted by $\langle s, s' \rangle$. If the second step's inequality holds, then the operation op is called *admissible* for S. This algorithm terminates in polynomial time, since each swap is performed in polynomial time, the number of swaps being performed is

$$\frac{log(cost(S_0)/cost(S^*))}{log\frac{1}{1-\epsilon/p(n,m))}},$$

where S_0 and S^* are the initial and optimum solutions respectively, and $log(cost(S_0))$ is polynomial in the input size.

When there are no admissible operations then we know that every operation reduces the cost by a factor of at most $\varepsilon/p(n, m)$, i.e. $cost(op(S)) \geq (1 - \frac{\epsilon}{p(n,m)})cost(S)$. To simplify the exposition, the assumption $cost(op(S)) \geq cost(S)$ is used. So, by adding at most $p(n, m)$ of such inequalities we can conclude that $cost(S) \leq \alpha \cdot cost(S^*)$ for some $\alpha \geq 1$, that is a *locality gap* α. Adding the corresponding original inequalities implies that $cost(S) \leq \alpha(1 + \epsilon)cost(S^*)$, that is an $\alpha(1 + \epsilon)$-approximation.

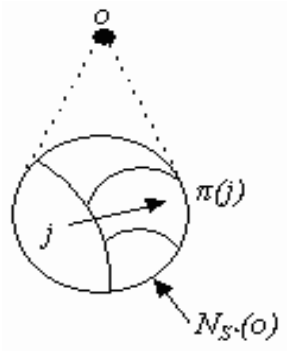

Figure 3: A matching π on $N_{S^*}(o)$

The following notation is used. Let s_j and o_j denote the service costs of a client j in the solutions S and S^* respectively. Let $N_S(s)$ denote the set of clients in C that are served by a facility $s \in S$ in the solution S. Similarly $N_{S^*}(o)$ denotes the set of clients in C that are served by a facility $o \in S^*$ in the solution S^*. Finally, for a subset $A \subseteq S$, let $N_S(A) = \cup_{s \in A} N_S(s)$.

Now we are ready to show that the local search procedure as defined above has a locality gap of 5. From the local optimality of S, we know that any swap $\langle s, o \rangle$ for $s \in S$ and $o \in S^*$,

$$cost(S - s + o) \geq cost(S) \text{ for all } s \in S, o \in S^*. \tag{4}$$

Combining these inequalities we can show that $cost(S) \leq 5 \cdot cost(S^*)$. Note that the algorithm and its analysis extend simply to the case when the clients $j \in C$ have arbitrary demands $d_{ij} \geq 0$ to be served. Also, the extension of this neighborhood to a p-Opt, where up to p facilities can be swapped simultaneously, has a $3 + 2/p$ locality gap, which is tight (see [46]). So, we have

Theorem 7.8 *A local search procedure for the metric k-median problem with operations defined as $op(S) := S - s + s'$ for $s \in S$ and $s' \notin S$, has a locality gap at most 5.*

Proof. Consider a facility $o \in S^*$. We partition $N_{S^*}(o)$ into subsets $p_s = N_{S^*}(o) \cap N_S(s)$ for $s \in S$. Consider a 1-1 and onto mapping $\pi : N_{S^*}(o) \rightarrow N_{S^*}(o)$ with the following property: for all $s \in S$ such that, $|p_s| \leq \frac{1}{2}|N_{S^*}(o)|$, we have, $\pi(p_s) \cap p_s = \varnothing$. It is easy to see that such a mapping π exists, (Fig. 3).

We say that a facility $o \in S^*$ is *captured* by a facility $s \in S$ if s serves more than half of the clients served by o, that is, $|N_S(s) \cap N_{S^*}(o)| > \frac{1}{2}|N_{S^*}(o)|$. Note that a facility $o \in S^*$ is captured by at most one $s \in S$. We call facility $s \in S$ *bad* if it captures some facility in S^* and *good* otherwise.

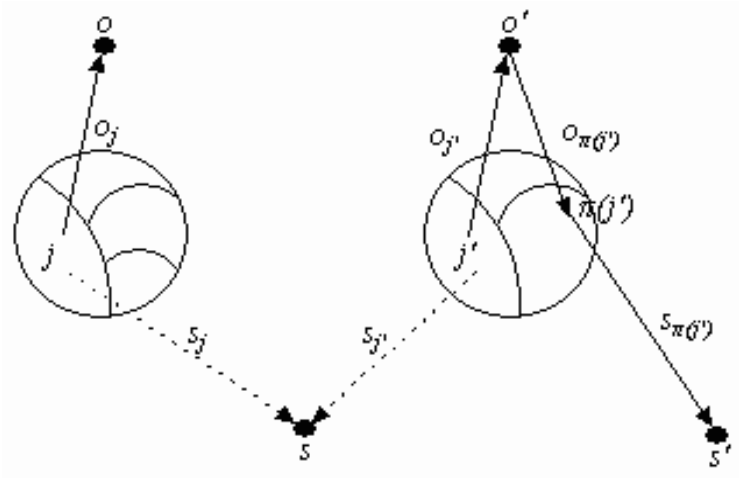

Figure 4: Reassigning the clients in $N_S(s) \cup N_{S^*}(o)$

We now consider k swaps, one for each facility in S^*. If some bad facility $s \in S$ captures exactly one facility $o \in S^*$ then we consider the swap $\langle s, o \rangle$. Suppose l facilities in S (and hence l facilities in S^*) are not considered in such swaps. These l facilities in S are either good or bad, and the bad facilities capture at least two facilities in S^*. Hence, there are at least $l/2$ good facilities in S. Now, consider l swaps in which the remaining l facilities in S^* get swapped with the good facilities in S such that each good facility is swapped-out at most twice.

It is easy to verify that the swaps considered above satisfy the following properties:

1. Each $o \in S^*$ is swapped-in exactly once.

2. Each $s \in S$ is swapped out at most twice. This is because a facility in S that captures more than one facility in S^* is never swapped-out and a facility that capture exactly one facility in S^* is swapped only with the facility that it captures.

3. If a swap $\langle s, o \rangle$ is considered, the facility s does not capture any facility $o' \neq o$.

We now analyze these swaps by considering an arbitrary swap $\langle s, o \rangle$. We place an upper bound on the increase in cost due to this swap by reassigning the clients in $N_S(s) \cup N_{S^*}(o)$ to the facilities in $S - s + o$ as follows (see Fig. 4). The clients $j \in N_{S^*}(o)$ are now assigned to o. Consider a client $j' \in N_S(s) \cap N_{S^*}(o')$, for $o' \neq o$. As s does not capture o', we have $|N_S(s) \cap N_{S^*}(o')| \leq \frac{1}{2}|N_{S^*}(o')$ and hence by the property of π, we have that $\pi(j') \notin N_S(s)$. Let $\pi(j') \in N_S(s')$. Note that the distance the client j' travels to the nearest facility in $S - s + o$ is at most $c_{j's'}$. Also from triangle inequality,

$c_{j's'} \leq c_{j'o} + c_{o\pi(j')} + c_{\pi(j')s'} = o_{j'} + o_{\pi(j')} + s_{\pi(j')}$. The remaining clients continue to be assigned to the old facilities. From inequality 4 we have,

$$cost(S - s + o) - cost(S) \geq 0.$$

Therefore,

$$\sum_{j \in N_{S^*}(o)} (o_j - s_j) + \sum_{j \in N_S(s), j \notin N_{S^*}(o)} (o_j + o_{\pi(j)} + s_{\pi(j)} - s_j) \geq 0 \tag{5}$$

As each facility $o \in S^*$ is swapped-in exactly once, the first term of the inequality 5 added over all the k swaps gives exactly $cost(S^*) - cost(S)$. For the second term, we use the fact that each s is swapped-out at most twice. Also for any $j \in C$, as s_j is the shortest distance from j to a facility in S, we get, using triangle inequality, $o_j + o_{\pi(j)} + s_{\pi(j)} \geq s_j$. Thus the second term of the inequality 5 added over all the k swaps is not greater than $2\sum_{j \in C}(o_j + o_{\pi(j)} + s_{\pi(j)} - s_j)$. But as π is 1-1 and onto mapping, $\sum_{j \in C} o_j = \sum_{j \in C} o_{\pi(j)} = cost(S^*)$ and $\sum_{j \in C}(s_{\pi(j)} - s_j) = 0$. Thus, $2\sum_{j \in C}(o_j + o_{\pi(j)} + s_{\pi(j)} - s_j) = 4 \cdot cost(S^*)$. Combining the two terms we get $cost(S^*) - cost(S) + 4 \cdot cost(S^*) \geq 0$. □

7.6.2 Capacity Allocation Problem

The capacity allocation problem (CAP) is a multi-commodity generalization of the single-commodity k-median problem, involving multiple types of service and the requirement that all nodes receive all these types from the corresponding supply nodes. This problem has applications in Internet content distribution. In [56], an exact algorithm that solves the problem, solving first a sufficient number of k-median problems, is presented. The combination of this algorithm with a polynomial time constant factor approximation algorithm for the k-median problem yields an approximation ratio for CAP as good as the one for the k-median. The extension of the algorithm, that we described above, to swaps of up to p facilities simultaneously, is the best known and has a $(3 + 2/p)(1 + \varepsilon)$-approximation ratio. Thus the CAP problem has a polynomial time $(3 + 2/p)(1 + \varepsilon)$-approximate algorithm.

7.6.3 k-means Clustering

In k-means clustering we are given a set of n data points in d-dimensional space $\mathcal{R}^d$ and an integer k, and the problem is to determine a set of k points in $\mathcal{R}^d$, called centers, to minimize the mean squared distance from each data point to its nearest center. For this problem no exact polynomial-time algorithm is known and although, asymptotically efficient algorithms exist (see [57]), they are not practical.

The main difference of this problem and the metric k-median, and thus the main difficulty of applying ones results on the other, is that in the first case the triangle inequality does not hold (however the doubled triangle inequality holds).

An iterative heuristic, called Lloyd's algorithm (see [58]), exists but it can converge to a local minimum arbitrarily worst than the global one. It starts with any feasible solution and then repeatedly computes the "neighborhood" of each center (the data points closest to it) and moves this center to the centroid of its "neighborhood".

In [59], a $(9 + \varepsilon)$-approximation local search algorithm is presented, based on the previous algorithm for k-median in [46], that we presented. It is based on swapping centers in and out of the solution set. This algorithm combined with the previous one has empirically shown a good practical performance.

7.7 Quadratic Assignment Problem

Given two $n \times n$ symmetric matrices $F = (f_{ij})$ and $D = (d_{ij})$, with a null diagonal, the symmetric Quadratic Assignment Problem (QAP) can be stated as follows:

$$\min_{\pi \in \Pi} \sum_{i=1}^{n} \sum_{k=i+1}^{n} f_{ik} d_{\pi(i)\pi(k)},$$

where Π is the set of all permutations of $\{1, 2, \ldots, n\}$. One of the major applications of the QAP is in location theory where f_{ij} is the flow of materials from facility i to facility j, and d_{ij} represents the distance from location i to location j. The objective is to find an assignment of all facilities to locations which minimizes the total cost.

The 2-exchange neighborhood is usually applied on this problem. That is, given a permutation $\pi = (\pi(1), \ldots, \pi(i), \ldots, \pi(j), \ldots, \pi(n))$, its neighbors are the $\frac{n(n-1)}{2}$ permutations of the form $\pi = (\pi(1), \ldots, \pi(j), \ldots, \pi(i), \ldots,$
$\pi(n))$ for $1 \leq i \leq j \leq n$, obtained from π by a swap.

QAP is NP-hard. Since Graph Partitioning under the swap neighborhood is a special case of the symmetric QAP under the 2-exchange neighborhood and as we have already seen it is PLS-complete, it follows that QAP under 2-exchange is PLS-complete, too.

In [60], a result has been obtained for QAP, which also applies to symmetric Travelling Salesman Problem, Graph Partitioning, k-Densest Subgraph, k-Lightest Subgraph and Maximum Independent Set, as they are subcases of QAP.

At first we give some notation. Let $s(A)$ denote the sum of all terms of a given matrix A. Let x and y two vectors of the same dimension. The maximum (resp. minimum) scalar product of x and y is defined by: $\langle x, y \rangle_+ = max_{\pi \in \Pi} \langle x, \pi y \rangle$ (resp. $\langle x, y \rangle_- = min_{\pi \in \Pi} \langle x, \pi y \rangle$). Let F_k and D_k denote the sum over the kth column of F and D, respectively. Let $\langle F, D \rangle_+$ (resp. $\langle F, D \rangle_-$) be an abbreviation for $\langle (F_1, \ldots, F_n), (D_1, \ldots, D_n) \rangle_+$

(resp. $\langle (F_1, \ldots, F_n), (D_1, \ldots, D_n) \rangle_-$). The following two theorems and the corollary have been proved:

Theorem 7.9 *For the QAP, let C^-_{loc} the cost of any solution found by a deepest descent local search*[4] *with the 2-exchange neighborhood, then the following inequality holds:*

$$C^-_{loc} \leq \frac{\langle F, D \rangle_-}{s(F)s(D)} n C_{AV},$$

where C_{AV} is the average cost of all possible permutations.

Corollary 7.1 *For the QAP, the following inequality holds:*

$$C^-_{loc} \leq \frac{n}{2} C_{AV}.$$

Moreover, there is a sequence of instances for which the ratio $C_{max}/\frac{n}{2}C_{AV}$ tends to infinity, where C_{max} is the maximum cost over all permutations.

Theorem 7.10 *If the matrices F and D are positive integer ones, a deepest descent local search will reach a solution with a cost less than $\frac{n}{2}C_{AV}$ in at most $O(nlog(\frac{s(F)s(D)}{2})$ iterations.*

Notice that, when one of the matrices, say F, has constant row sums, i.e. $Fe = \lambda e$, for e a vector of all ones, then $\langle F, D \rangle_+ / s(F)s(D) = 1/n$ and it follows that $C^-_{loc} \leq C_{AV}$ from theorem 7.9. Now, let us see the applications of the above theorems on some known problems.

7.7.1 The Symmetric Travelling Salesman Problem

This problem can be seen as a particular case of QAP by considering D to be the distance matrix and F to be defined by $f_{i,i+1} = f_{i+1,i} = 1$ with $1 \leq i \leq n-1, f_{n,1} = f_{1,n} = 1$ and $f_{ij} = 0$ otherwise. Using the above remark, it is obtained, $C^-_{loc} \leq C_{AV}$, for the 2-exchange neighborhood, which is the same as 2-Opt.

[4] That is, the local search heuristic which successively replaces the current solution by the *best* neighboring one.

7.7.2 The unweighted Graph Partitioning Problem

Recall that in this problem we are given a graph and we have to partition its vertices in two equal-sized subsets A and B, such that the number of edges having one extremity in A and the other in B, is minimized. For this problem, D is the adjacency matrix of the graph, and

$$F = \begin{pmatrix} 0 & U \\ U & 0 \end{pmatrix}$$

where U is the $n/2 \times n/2$ matrix, with $u_{ij} = 1, i, j = 1, \ldots, n/2$. Using the above remark, it is obtained for the swap neighborhood, $C^{-}_{loc} \leq C_{AV}$.

7.7.3 The unweighted k-Lightest and the k-Densest Subgraph Problems

These problems are defined as follows. Given a graph $G = (V, E)$ and a number $m(m \leq |V|)$, find m vertices of G such that the number of edges in the subgraph induced by these vertices is minimum (respectively maximum). These problems have also been studied in [60], but they were referred to as Generalized Maximum independent Set and Generalized Maximum Clique Problems. They can be modelized by a QAP with D the adjacency matrix of graph G and $F = (f_{ij})$, where $f_{ij} = 1$ if $i \neq j, 1 \leq i, j, \leq m$ and $f_{ij} = 0$ otherwise. In the sequel it is considered that $d_1, d_2, \ldots, d_n$ are the degrees of the vertices of G arranged in decreasing order. The following result was obtained.

Proposition 7.1 *The local optimal solution found by a deepest local search with the swap neighborhood satisfies* $C^{-}_{loc} \leq ((m-1)/2(n-1))(d_1 + d_2 + \ldots + d_m)$ *(respectively* $C^{+}_{loc} \geq ((m-1)/2(n-1))(d_n + d_{n-1} + \ldots + d_{n-m+1})$*) for the minimization (respectively the maximization problem).*

7.7.4 The Maximum Independent Set Problem

Finally for MIS we have the following proposition

Proposition 7.2 *If* $d_1 + d_2 + \ldots + d_k \leq \lfloor 2c(n-1)/(k-1) \rfloor_*$, *with* $2 \leq k \leq n$ *and* c *any integer, the deepest local search with the swap neighborhood finds an independent set with at least* $k - c + 1$ *vertices. By definition,* $\lfloor x \rfloor_*$ *is equal to* $x - 1$ *if* x *is an integer, and* $\lfloor x \rfloor$ *otherwise.*

8 Conclusions and open problems

Local Search is a method extensively used to approximately solve NP-hard Combinatorial Optimization Problems. The aim of this work was to sum up some main theoretical results that we have for Local Search. So, at first, we saw the theory of PLS-completeness, which gives us the instruments to recognize the difficulty of Local Search Problems and we concluded that for the PLS-complete problems the standard local search heuristic takes exponential time in the worst case.

Then, we saw that the quality of the local optima depends on the NP-hardness of the corresponding optimization problems. Theorem 6.1 provides negative indications for the approximation efficiency of local search on NP-hard optimization problems. However, apart from the experimentally observed power of local search and the probabilistic verification of this ability, there are lately, a lot of results, which provide ε-approximation guarantees of local search for many common NP-hard problems. Section 7 presents these results giving also some characteristic proofs. All the problems mentioned there, are either unweighted or with polynomially bounded weights subcases of their general problems. Since local search is pseudopolynomial, the standard local search heuristic terminates in a rather competitive polynomial time for them.

There are two more theoretical questions concerning local search. The first one is about the parallel complexity of determining if we are on a local optimum solution and computing a better neighbor if we are not. Generally this problem is independent of the difficulty of the local search problem itself. Hence, for Max-Sat/Flip the complexity is in NC while Graph Partitioning/KL is P-complete (see [4]), both problems being PLS-complete. The second question asks about the complexity of the standard local optimum problem. That is, for a given problem, starting from a specific solution, how fast we can find the local optimum that the standard local search heuristic would have produced. It turns out, that for all PLS-complete problems this latter problem is PSPACE-complete (see [4]).

Closing this work, it would be a great lack not to refer to some other uses of local search. First of all, due to its efficiency and simplicity, local search methods are also used to solve polynomial problems, such as Linear Programming, Maximum Matching and Maximum Flow. The well-known algorithm Simplex, is a local search heuristic, which explores an exact neighborhood (each time, the adjacent vertices of the current vertice on the polytope) and it is proved to take exponential time in the worst case for many pivoting rules. Simplex is used to solve Continuous Linear Programming Problems, despite the existence of polynomial algorithms, such as the interior point algorithms, because it works much better in practice.

Another use of local search is as a tool in proofs of existence of a solution to a problem, i.e. that a search problem is total. For example, we saw in subsection 3.6 that the

proof, that there is always a stable configuration in neural networks of the Hopfield model, depends on proving that the initial search problem can be transformed into a local search problem, under appropriate cost and neighborhood functions, and hence the existence of the stable configurations are guaranteed by the existence of the local optima. In [4], there is the Submatrix problem proposed by Knuth, for which the same argument guarantees the existence of its solution.

Recently, in [61], a connection between PLS Theory and Game Theory established. More particularly, the problems of finding *pure Nash equilibria* in General Congestion Games, Symmetric Congestion Games and Asymmetric Network Congestion Games were shown to be PLS-complete. The reductions follow from the Pos NAE-3Sat/Flip. The use of local search in proofs of existence, referred in the previous paragraph, is also applied to proofs of existence of pure Nash equilibria in games, called as the potential function method. Let's call a game as a *general potential function game* if there is a function ϕ such that for any edge of the Nash dynamics graph (s, s') with defector i we have $sgn(\phi(s') - \phi(s)) = sgn(u_i(s') - u_i(s))$. These games, obviously have pure Nash equilibria, from the potential function argument. An interesting result, presented in [61], is a converse one, that the class of general potential games essentially comprises all of PLS.

An extension of local search, proposed in [18] as a general paradigm useful for developing simple yet efficient approximation algorithms, is *non-oblivious local search.* Non-oblivious local search allows the cost function, used to find a local optimum, to be different from that of the original problem, hence the search can be directed to better quality solutions. It is showed that every MAX-SNP problem can be approximated to within constant factors by this method. Ausiello and Protasi defined, in [62], the class GLO (guaranteed local optima) of combinatorial optimization problems which have the property that for all locally optimum solutions, the ratio between the value of the global and the local optimum is bounded by a constant. Vertex Covering does not belong to GLO but it is MAX-SNP, hence GLO is a strict subset of non-oblivious GLO.

Furthermore, local search is the base of most *metaheuristics*. In [63], there is an overview on metaheuristics and the following definition of Stützle, [64], is given among others: "Metaheuristics are typically high-level strategies which guide an underlying, more problem specific heuristic, to increase their performance. The main goal is to avoid the disadvantages of iterative improvement and, in particular, multiple descent by allowing the local search to escape from local optima. This is achieved by either allowing worsening moves or generating new starting solutions for the local search in a more "intelligent" way than just providing random initial solutions". Some of the most common metaheuristics are Tabu Search, Simulated Annealing, GRASP, Iterated Local Search, Variable Neighborhood Search, Evolutionary Computation and Ant Colony Optimization. Apart from the last one, all the other methods are variations of local search, in which the cost function or the neighborhood or the allowed perturbations on a solution change

dynamically during the search. The performance of such methods is studied empirically and it would be rather difficult to have some purely theoretical results about them.

Finally, there is the *landscape theory*, which tries to theoretically justify why a neighborhood is better in practice than another for a given optimization problem. The ruggedness of the landscape which is formed by the cost function and the neighborhood is measured, since a good agreement between ruggedness and difficulty for local search is observed. A hierarchy of the combinatorial optimization problems can be obtained relatively to their ruggedness. More about this field one can find in [65, 66].

There are a lot of theoretical and experimental *open problems* in the area of local search. First of all, we do not know the exact relation of the class PLS with the classes P and NP. There are, also, many interesting local search problems, such as TSP/2-Opt, that we do not know if they are PLS-complete or not. Furthermore, a lot of NP-hard problems have approximately been solved, within constant factors, by local search methods, so improvement of these factors and extension of such results to more problems, would be of great importance. On the experimental aspect of research of this field, the average performance of local search algorithms both in computational time and in approximation efficiency, is rather interesting. In landscape theory there are unanswered questions about the definition of the ruggedness and other characteristics of the landscapes of local search problems. Neighborhoods of exponential size are under research too, in order to examine whether we can search them efficiently or if we can guarantee ε-local optima with them. Finally, the recent use of local search methods in game theory seems to give some first, very interesting results.

References

[1] D.S. Johnson, C.H. Papadimitriou & M. Yannakakis. *How easy is local search?* Journal of Computer and System Sciences (1988), **37(1)**, pp. 79-100.

[2] B.W. Kernighan & S. Lin. *An efficient heuristic procedure for partioning graphs.* Bell System Technical Journal (1970), **49**, pp. 291-307.

[3] P. Christopoulos. *Local Search and PLS-completeness.* Undergraduate Thesis, Department of Informatics and Telecommunications, University of Athens (2003).

[4] M. Yannakakis. *Computational Complexity.* In E. Aarts and J. k. Lenstra, *Local Search in Combinatorial Optimization*, Wiley-Interscience Publication (1998), Chapter 2, pp. 19-55.

[5] C.H. Papadimitriou. *The complexity of the Lin-Kernighan heuristic for the TSP.* SIAM Journal on Computing (1992), **21**, pp. 450-465.

[6] J.J. Hopfield. *Neural networks and physical systems with emergent collective computational abilities.* Proceedings of the National Academy of Sciences of the USA (1982), **79**, pp. 2554-2558.

[7] J. Bruck & J.W. Goodman, *A generalized convergence theorem for neural networks*, IEEE Transactions on Information Theory (1988) **34**, pp. 1089-1092.

[8] G. Godbeer, *On the computational complexity of the stable configuration problem for the connectionist models*, MSc thesis (1987), Department of Computer Science, University of Toronto.

[9] J. Lipscomb, *On the computational complexity of finding a connectionist model's stable state of vectors*, MSc thesis (1987), Department of Computer Science, University of Toronto.

[10] A.A. Schaffer & M. Yannakakis. *Simple local search problems that are hard to solve.* SIAM Journal on Computing (1991), **20(1)**, pp. 56-87.

[11] M.W. Krentel. *Structure in locally optimal solutions.* 30th Annual Symposium on Foundations of Computer Science (1989), IEEE Computer Society Press, Los Alamitos, CA, pp. 216-222.

[12] M.W. Krentel. *On finding and verifying locally optimal solutions.* SIAM J. on Computing (1990), **19**, pp. 742-751.

[13] J.B. Orlin, A.P. Punnen, A.S. Schulz. *Approximate Local Search In Combinatorial Optimization.* (July 2003), MIT Sloan Working Paper No. 4325-03.

[14] N. Christofides. *Worst-case analysis of a new heuristic for the travelling salesman problem.* Report 388, Graduate School of Industrial Administration, Carnegie Mellon University, Pittsburgh, PA, (1976).

[15] C.H. Papadimitriou. *On selecting a satisfying truth assignment.* In Proceedings of the 32nd Annual IEEE Symposium on Foundations of Computer Science (1991), FOCS'91, pp. 163-169.

[16] E. Dantsin, A. Goerdt, E.A. Hirsch, R. Kannan, J. Kleinberg, C. Papadimitriou, P. Raghavan and U. Schöning. *A deterministic* $(2 - \frac{2}{k+1})^n$ *algorithm for* k*-SAT based on local search.* (2001), http://citeseer.nj.nec.com/dantsin01deterministic.html.

[17] P. Hansen and B. Jaumard. *Algorithms for the maximum satisfiability problem.* Computing (1990), **44**, pp. 279-303.

[18] S. Khanna, R. Motwani, M. Sudan, U. Vazirani. *On syntactic versus computational views of approximability.* Technical Report TR95-023, Ellectronic colloquium on computational complexity (1995), http://www.eccc.uni-trier.de/eccc/.

[19] R. Battiti and M. Protasi. *Solving MAX-SAT with non-oblivious functions and history-based heuristics.* In *Satisfiability problems: Theory and applications,* DIMACS: Series in discrete mathematics and theoretical computer science (1997), no. 35, AMS and ACM Press.

[20] J. Håstad. *Some optimal inapproximability results.* Proceedings of the 29th ACM Symposium on the Theory of Computation (1997), ed. L. Longpré, ACM, New York, pp. 1-10.

[21] F. Alizadeh. *Optimization over the positive semi-definite cone: interior point methods and combinatorial applications.* P.M. Pardalos (Ed.), Advances in Optimization and Parallel Computing, North-Holland, Amsterdam, Netherlands, (1992), pp. 1-25.

[22] C. Helmberg, F. Rendl, R.J. Vanderbei, H. Wolkowicz. *An interior point method for semidefinite programming.* Technical Report (1994), University of Graz.

[23] G. Ausiello. *Compexity and approximation: Combinatorial optimization problems and their approximation properties.* Springer (1999).

[24] A. Bertoni, P. Campadelli, G. Grossi. *An approximation algorithm for the maximum cut problem and its experimental analysis.* Discrete Applied Mathematics (2001), **110**, pp. 3-12.

[25] C.H. Papadimitriou & K. Steiglitz. *On the complexity of local search for the TSP.* SIAM Journal of Computing (1977), **6**, pp. 76-83.

[26] S. Sahni & T. Gonzales. *P-complete approximation problems.* Journal of the Association for Computing Machinery (1976), **23**, pp. 555-565.

[27] B. Chandra, H. Karloff, C.A. Tovey. *New results on the old k-opt algorithm fot the TSP.* SIAM Journal on Computing (1999), **28(6)**, pp. 1998-2029.

[28] M. Halldórsson. *Approximating discrete collections via local improvements.* Proceedings of the Sixth Annual ACM-SIAM Symposium on Discrete Algorithms (1995), ACM New York and SIAM Philadelphia PA, pp. 160-169.

[29] B. Chandra and M.M. Halldórsson. *Greedy local improvement and weighted packing approximation.* In SODA (1999), pp. 169-176.

[30] B. Chandra and M.M. Halldórsson. *Greedy local improvement and weighted packing approximation.* Journal of Algorithms (2001), **39(2)**, pp. 223-240.

[31] P. Berman. *A $d/2$ approximation for maximum weight independent set in d-claw free graphs.* Nordic Journal of Computing (2000), **7(3)**, pp. 178-184.

[32] R. Duh and M. Fürer. *Approximation of k-set cover by semi-local optimization.* ACM Symposium on Theory of Computing (1997), pp. 256-264.

[33] S. Khuller, R. Bhatia, R. Pless. *On local search and placement of meters in networks.* Symposium on Discrete Algorithms (2000), pp. 319-328.

[34] J. Kleinberg, É. Tardos. *Approximation algorithms for classification problems with pairwise relationships: Metric labeling and Markov random fields.* In Proceedings of the 40th Annual IEEE Symposium on Foundations of Computer Science (1999), pp. 14-23.

[35] J. Besag. *On the statistical analysis of dirty pictures.* J. Royal Statistical Society B (1986), **48(3)**, pp. 259-302.

[36] D. Greig, B.T. Porteous, A. Seheult. *Exact maximum a posteriori estimation for binary images.* J. Royal Statistical Society B (1989), **51(2)**, pp. 271-279.

[37] Y. Boyjov, O. Veksler, R. Zabih. *Markov random fields with efficient approximations.* In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, IEEE computer Society Press (1998), Los Alamitos, Calif, pp. 648-655.

[38] O. Veksler. *Efficient graph-based energy minimization methods in computer vision.* PhD Thesis (1999), Department of Computer Science, Cornell University.

[39] H. Ishikawa, D. Geiger. *Segmentation by grouping junctions.* In Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (1999), pp. 125-131.

[40] A. Karzanov. *Minimum 0-extension of graph metrics.* Europ. J. Combinat. (1998), **19**, pp. 71-101.

[41] A. Karzanov. *A combinatorial algorithm for the minimum $(2, r)$-metric problem and some generilizations.* Combinatorica (1999), **18(4)**, pp. 549-569.

[42] C. Chekuri, S. Khanna, J. Naor, L. Zosin. *Approximation algorithms for the metric labeling problem via a new linear programming formulation.* Symposium on Discrete Algorithms (2001), pp. 109-118.

[43] A. Gupta, É. Tardos. *A constant factor approximation algorithm for a class of classification problems.* In Proceedings of the 32nd Annual ACM Symposium on the Theory of Computating (2000), pp. 652-658

[44] D.B. Shmoys, É. Tardos, K. Aardal. *Approximation algorithms for facility location problems.* In Proceedings of the 29th Annual ACM Symposium on Theory of Computing (1997), pp. 265-274.

[45] M. Korupolu, C. Plaxton, R. Rajaraman. *Analysis of a local search heuristic for facility location problems.* Technical Report 98-30, DIMACS, June 1998.

[46] V. Arya, N. Garg, R. Khandekar, A. Meyerson, K. Mungala, V. Pandit. *Local search heuristic for k-median and facility location problems.* Proceedings of the 33rd Annual ACM Symposium on the Theory of Computing (2001), pp. 21-29.

[47] M. Charikar, C. Chekuri, A. Goel, S. Guha. *Rounding via trees: Deterministic approximation algorithms for group steiner trees and k-median.* In Proceedings of the 30th Annual ACM Symposium on Theory of Computing (1998), pp. 106-113.

[48] J.H. Lin, J.S. Vitter. *ε-approximations with minimum packing constraint violation.* In Proceedings of the 24th Annual ACM Symposium on Theory of Computing (1992), pp. 771-782.

[49] J.H. Lin, J.S. Vitter. *Approximation algorithms for geometric median problems.* Information Processing Letters (1992), **44**, pp. 245-249.

[50] S. Arora, P. Raghavan, S. Rao. *Approximation schemes for Euclidean k-medians and related problems.* In Proceedings of the 30th Annual ACM Symposium on Theory of Computing (1998), pp. 106-113.

[51] D.S. Hochbaum. *Heuristics for the fixed cost median problem.* Mathematical Programming (1982), **22**, pp. 148-162.

[52] F.A. Chudak. *Improved approximation algorithms for the uncapacitated facility location problem.* In Proceedings of the 6th Conference on Integer Programming and Combinatorial Optimization (1998). pp. 180-194.

[53] F.A. Chudak and D.P. Williamson. *Improved approximation algorithms for capacitated facility location problems.* Proceedings of the 7th International IPCO Conference (1999).

[54] F. Chudak and D.B. Schmoys. *Improved approximation algorithms for a capacitated facility location problem.* In Proceedings of the 10th Annual ACM-SIAM Symposium on Discrete Algorithms (1999), pp. 875-876

[55] M. Pál, É. Tardos and T. Wexler. *Facility location with nonuniform hard capacities.* IEEE Symposium on Foundations of Computer Science (2001), pp. 329-338.

[56] N. Laoutaris, V. Zissimopoulos, I. Stavrakakis. *Joint object placement and node dimensioning for Internet content distribution.* Information Processing Letters, to appear.

[57] J. Matoušek. *On approximate geometric k-clustering.* Discrete and Computational Geometry (2000), **24**, pp. 61-84.

[58] Q. Du, V. Faber, M. Gunzburger. *Centroidal Voronoi tesselations: Applications and algorithms.* SIAM review (1999), **41**, pp. 637-676.

[59] T. Kanungo, D.M. Mount, N.S. Netanyahu, C.D. Piatko, R. Silverman, A.Y. Wu. *A local search approximation algorithm for k-means clustering.* In Proceedings of the 18th Annual Symposium on Computational Geometry (2002), Barcelona, Spain, pp. 10-18.

[60] E. Angel, V. Zissimopoulos. *On the quality of local search for the quadratic assignment problem.* Discrete Applied Mathematics (1998), **82**, pp. 15-25.

[61] A. Fabrikant, C. Papadimitriou, K. Talwar. *The complexity of pure Nash equilibria.* Papadimitriou's home page (2003), http://www.cs.berkeley.edu/ christos.

[62] G. Ausiello, M. Protasi. *Local search, reducibility and approximability of NP optimization problems.* Inform. Process. Lett. (1995), **54**, pp. 73-79.

[63] C. Blum, A. Roli. *Metaheuristics in Combinatorial Optimization: Overview and Conceptual Comparison.* ACM Computing Surveys (2003), **35:3**, pp. 268-308.

[64] T. Stützle. *Local search algorithms for combinatorial problems - analysis, algorithms and new applications.* DISKI - Dissertationen zur Künstliken Intelligenz. infix, Sankt Augustin, Germany.

[65] E. Angel, V. Zissimopoulos. *On the classification of NP-complete problems in terms of their correlation coefficient.* Discrete Applied Mathematics (2000), **99**, pp. 261-277.

[66] C.M. Reidys, P.F. Stadler. *Combinatorial Landscapes.* SIAM Review (2002), **44**, pp. 3-54.