

A Selection of Useful Theoretical Tools for the Design and Analysis of Optimization Heuristics

*G. Gutin** *D. Karapetyan*†

Abstract

An intensive practical experimentation is certainly required for the purpose of heuristics design and evaluation, however a theoretical approach is also important in this area of research. This paper gives a brief description of a selection of theoretical tools that can be used for designing and analyzing various heuristics. For design and evaluation, we consider several examples of preprocessing procedures and probabilistic instance analysis methods. We also discuss some attempts at the theoretical explanation of successes and failures of certain heuristics.

1 Introduction

While optimization heuristics do not offer any performance guarantees in general, in practice they often produce better solutions than approximation and other algorithms. However, many heuristic methods lack theoretical foundations and rely essentially on computational experiments which often cannot explain good and bad behavior of heuristics. Many researchers designing and analyzing optimization heuristics realize that development of theoretical foundations of heuristics is very important and it is the greatest challenge of future research of heuristics. In this relatively short paper we give brief descriptions of some theory that can be used for designing and analyzing various heuristics. We also provide some links for further reading on the topics we discuss.

Certainly, we do not cover all useful theory and the choices of the topics reflect, at least partially, our personal research interests. However, our sample of topics covers various aspects of theory and includes several recent developments not widely known yet. Thus, we believe that this short overview contains information that is of interest to many researchers developing and analyzing heuristics for various optimization problems.

Section 2 is devoted to preprocessing, which is extremely important in developing very efficient heuristics. Preprocessing allows us to significantly speed up optimization heuristics and sometimes transform virtually unsolvable instances of optimization problems into instances that can be solved in a short period of time. We describe

*Department of Computer Science, Royal Holloway University of London, Egham, Surrey TW20 0EX, UK, G.Gutin@rhul.ac.uk

†Department of Computer Science, Royal Holloway University of London, Egham, Surrey TW20 0EX, UK, Daniel.Karapetyan@gmail.com

some successful examples of preprocessing and provide a short introduction into kernelization, which is a more systematic way of developing preprocessing algorithms for some optimization problems.

In Section 3, we discuss probabilistic analysis of testbeds for computational experiments. Probabilistic approach sometimes allows us to provide a good estimate for the optimal solution when all other approaches fail. Section 3 includes very recent results on multidimensional assignment problem and better known results on the traveling salesman problem.

In Section 4, we provide a theoretical explanation of why certain greedy heuristics often obtain poor quality solutions. Our explanation is based on domination analysis, which is a recent alternative and complement to approximation analysis.

Local search is an important part of memetic algorithms, see, e.g., [38, 39, 45]. Section 5 is devoted to Very Large Neighborhood Local Search (VLNLS), a local search approach whose origins can be traced to 1980s (or even earlier in a wider interpretation), but who was mainly investigated in the last decade. We show that domination analysis offers a good explanation of why some VLNLS heuristics are inferior to well-known local search heuristics based on small neighborhoods.

While designing and analyzing heuristics, two or more sections of this paper may be useful. However, the sections are mainly independent and can be read separately.

2 Preprocessing

Different preprocessing procedures are often used for hard problems to reduce the computation time. There are examples of such approaches in integer and linear programming (e.g., [34, 47]) as well as for the Vehicle Routing Problem [41]. In some cases preprocessing plays the key role in an algorithm (e.g., [17]). Next we show some examples, that we find particularly interesting, of efficient preprocessing procedures for several problems.

2.1 Preprocessing in Linear Programming

One of the most striking success stories of preprocessing is the one of CPLEX simplex algorithms for linear programming (LP). In Example 3 of [7] Bixby considers a very large LP problem that can be solved by a CPLEX LP solver in half an hour after preprocessing, but cannot be solved after 4 hours without preprocessing. Preprocessing for LP problems combines various approaches, and one of the most useful of them is exploitation of network structures. In the rest of this subsection, we describe a preprocessing method for LP problems close to networks (which is often the case for real-world LP problems) introduced in [23]. Gulpinar, Gutin, Mitra and Zverovitch [23] showed the efficiency of the method in computational experiments.

We consider an LP problem in the standard form stated as

$$\text{Minimize } \{p^T x; \text{ subject to } Ax = b, x \geq 0\}.$$

LP problems have a number of equivalent, in a sense, forms that can be obtained from each other by various operations. Often scaling operations, that is multiplications of rows and columns of the matrix A of constraints by non-zero constants, are applied such that A contains a large number of entries equal 0, +1 or -1 .

A matrix B is a *network (matrix)* if B is a $(0, \pm 1)$ -matrix (that is, entries of B belong to the set $\{1, 0, -1\}$) and every column of B has at most one entry equal to 1 and at most one entry equal to -1 . The operation of *reflection* of a row of a matrix B

changes the signs of all non-zero entries of this row. A matrix B is a *reflected network (matrix)* if there is a sequence of row reflections that transforms B into a network matrix. The *problem of detecting a maximum embedded reflected network* (DMERN) is to find the maximum number of rows that form a submatrix B of A such that B is a reflected network. This number is denoted by $\nu(A)$. The DMERN problem is known to be NP-hard [4] and Gulpinar et al. [23] designed a heuristic that solves the DMERN problem to optimality when A is a reflected network. In what follows we assume that A is a $(0, \pm 1)$ -matrix as we can delete all rows of A containing entries other than 0, +1 or -1 without any effect on the solution of the DMERN problem.

The heuristic uses the notion of a signed graph, i.e., a graph G in which each edge has sign + or -. For a signed graph G , G^- denotes the subgraph of G whose vertices are the same as in G and whose edges are the negative edges in G . For a non-empty subset W of the vertex set $V(G)$ of graph G , the W -switch of G is the signed graph G^W obtained from G by changing the signs of the edges between W and $V(G) \setminus W$. For a $(0, \pm 1)$ -matrix $A = [a_{ik}]$ with n rows, we construct a signed graph $G(A)$ as follows: the vertex set of $G(A)$ is $\{1, 2, \dots, n\}$; $G(A)$ has a positive (negative) edge ij if and only if $a_{ik} = -a_{jk} \neq 0$ ($a_{ik} = a_{jk} \neq 0$) for some k . The graph $G(A)$ may have parallel edges of opposite signs. A set I of vertices of a graph H is *independent* if no pair of vertices in I are adjacent; $\alpha(H)$ denotes the cardinality of a maximum independent set of vertices in H , the *independence number* of H .

The following is the main result of [23]. This theorem allows us to consider signed graphs rather than matrices.

Theorem 1. *For a $(0, \pm 1)$ -matrix A , we have $\nu(A) = \max\{\alpha((G^W)^-): W \subseteq V\}$.*

We say that a graph H is a *network graph* if there is a reflected network matrix B such that $H = G(B)$.

Lemma 2. *Every signed tree T is a network graph.*

Proof. We prove the lemma by induction on the number of edges in T . The lemma is true when the number of edges is one. Let x be a vertex of T of degree one. By the induction hypothesis, there is a set $W \subseteq V(T) \setminus \{x\}$ such that $\mu((T - x)^W) = 0$. In T^W the edge e incident to x is positive or negative. In the first case, let $W' = W$ and the second case, let $W' = W \cup \{x\}$. Then, $\mu(T^{W'}) = 0$. \square

Now we will describe the heuristic of [23]. Gulpinar et al. [23] proved that this heuristic finds the optimal solution when A is a reflected network matrix.

1. Construct the signed graph $G = G(A)$.
2. Find a spanning forest T in G .
3. Using a recursive algorithm based on the proof of Lemma 2, compute $W \subseteq V$ such that $(T^W)^- = \emptyset$.
4. Using some algorithm, find a maximal independent set I in the graph $(G^W)^-$.

The vertices of I correspond to rows of A that form a reflected network matrix.

2.2 Preprocessing for the Generalized Traveling Salesman Problem

The *Generalized Traveling Salesman Problem* (GTSP) is defined as follows. We are given a weighted complete undirected graph G on N vertices and a partition $V =$

$V_1 \cup V_2 \cup \dots \cup V_M$ of its vertices; the subsets V_i are called *clusters*. The objective is to find a minimum weight cycle containing exactly one vertex from each cluster. Here we present a problem reduction algorithm that deletes redundant vertices and edges, preserving the value of the optimal solution. The algorithm's running time is $O(N^3)$ in the worst case, but it is significantly faster in practice. In our experiments the problem size was reduced by 15–20% on average and this has decreased the solution time by approximately 45% for each of the three solvers considered [28].

The weight of an edge xy of G is denoted by $\text{dist}(x, y)$ and is often called the *distance* between x and y . The expression $\text{cluster}(x)$ denotes cluster that contains the vertex x .

Definition 3. Let C be a cluster, $|C| > 1$. We say that a vertex $r \in C$ is *redundant* if, for each pair x, y of vertices from distinct clusters different from C , there exists $s \in C \setminus \{r\}$ such that $\text{dist}(x, s) + \text{dist}(s, y) \leq \text{dist}(x, r) + \text{dist}(r, y)$.

Testing this condition for every vertex takes approximately $O(N^3 \cdot |\overline{V}|)$ operations, where $|\overline{V}|$ is the average cluster size. In some cases it is possible to reduce significantly the preprocessing time.

Let us fix some vertex $r \in C$. For every $s \in C$ and every $x \notin C$ calculate the value $\Delta_x^{r,s} = \text{dist}(x, r) - \text{dist}(x, s)$. Observe that a vertex r is redundant if there is no pair of vertices $x, y \notin C$ from different clusters such that $\Delta_x^{r,s} + \Delta_y^{r,s} < 0$ for every s , i.e., r is redundant if for every $x, y \notin C$, $\text{cluster}(x) \neq \text{cluster}(y)$, there exists $s \in C \setminus \{r\}$ such that $\Delta_x^{r,s} + \Delta_y^{r,s} \geq 0$. That is due to $\Delta_x^{r,s} + \Delta_y^{r,s} = \text{dist}(x, r) - \text{dist}(x, s) + \text{dist}(y, r) - \text{dist}(y, s) = \text{dist}(x, r) + \text{dist}(r, y) - (\text{dist}(x, s) + \text{dist}(s, y))$.

There is a way to accelerate the algorithm. If

$$\min_{x \notin Z} \max_{s \in C} \Delta_x^{r,s} + \min_{x \in Z} \max_{s \in C} \Delta_x^{r,s} < 0$$

for some cluster Z , then r cannot be reduced immediately. We can use an equivalent condition:

$$\min_{x \in \bigcup_{j < i} V_j} \max_{s \in C} \Delta_x^{r,s} + \min_{x \in V_i} \max_{s \in C} \Delta_x^{r,s} < 0$$

This condition can be tested during the Δ values calculation by accumulating the value of $\min_{x \in \bigcup_{j < i} V_j} \max_{s \in C} \Delta_x^{r,s}$.

Removing a redundant vertex may cause a previously irredundant vertex to become redundant. Thus it is useful to check redundancy of vertices in cyclic order until we see that, in the last cycle, no vertices are found to be redundant. However, in the worst case, that would lead to the total number of the redundancy tests to be $\Theta(N^2)$. Our computational experience has shown that almost all redundant vertices will be found even if we restrict ourselves to testing each vertex of GTSP at most twice.

Definition 4. Let u, v be a pair of vertices from distinct clusters U and C respectively. Then the edge uv is *redundant* if for each vertex $x \in V \setminus (U \cup C)$ there exists $v' \in C \setminus \{v\}$ such that $\text{dist}(u, v') + \text{dist}(v', x) \leq \text{dist}(u, v) + \text{dist}(v, x)$.

An algorithm we introduce for edge reduction is as follows. Given a vertex $v \in C$, $|C| > 1$, we detect redundant edges incident with v using the following procedure:

1. Select an arbitrary vertex $v'' \in C \setminus \{v\}$.
2. Set $P_x = \Delta_x^{v, v''}$ for each vertex $x \in V \setminus C$.

3. Sort array P in non-decreasing order.
4. For each cluster $U \neq C$ and for each vertex $u \in U$ do the following:
 - (a) $\delta = \Delta_u^{v,v''}$
 - (b) For each item $\Delta_x^{v,v''}$ of the array P such that $\Delta_x^{v,v''} + \delta < 0$ check the following: if $x \notin U$ and $\Delta_x^{v,v'} + \Delta_u^{v,v'} < 0$ for every $v' \in C \setminus \{v, v''\}$, the edge uv is not redundant, continue with the next u .
 - (c) Edge uv is redundant, set $\text{dist}(u, v) = \infty$.

The procedure is executed exactly one time for every vertex v such that $|\text{cluster}(v)| > 1$. The whole edge reduction procedure takes $O(N^3)$ operations. The proof of the algorithm can be found in [28].

2.3 Kernelization

While the preprocessing methods and approaches described above are of heuristic nature and largely problem specific, there is a fast growing area of research where preprocessing is being developed in a solid theoretical framework. The area is *Fixed-Parameter Algorithmics (FPA)* and preprocessing there is called *kernelization*. Below we give a very brief introduction to FPA and kernelization; for detailed accounts of the area, see the monographs [13, 16, 42].

FPA is a relatively new approach for dealing with intractable computational problems. In the framework of FPA we introduce a parameter k , which is often a positive integer (but may be a vector, graph, or any other object for some problems) such that the problem at hand can be solved in time $O(f(k)n^c)$, where n is the size of the problem instance, c is a constant not dependent on n or k , and $f(k)$ is an arbitrary computable function not dependent on n . The ultimate goal is to obtain $f(k)$ and c such that for small or even moderate values of k the problem under consideration can be completely solved in a reasonable amount of time.

As an example, consider the *Vertex Cover problem (VC)*: given an undirected graph G (with n vertices and m edges), find a minimum number of vertices such that every edge is incident to at least one of these vertices. In the (naturally) parameterized version of VC, k -VC, given a graph G , we are to check whether G has a vertex cover with at most k vertices. k -VC admits an algorithm of running time $O(1.2738^k + kn)$ obtained in [11] that allows us to solve VC with k up to several hundreds. Without using FPA, we would be likely to end up with the obvious algorithm of complexity $O(mn^k)$. The last algorithm is far too slow even for small values of k such as $k = 10$.

Parameterized problems that admit algorithms of complexity $O(f(k)n^c)$ are called *fixed-parameter tractable (FPT)*. Notice that not every parameterized problem is FPT, but there are many problems that are FPT. A parameterized problem is FPT if and only if it admits kernelization, which is defined as follows. For a parameterized decision problem Π given by pairs (I, k) , where I is an instance of Π and k is the parameter, a *kernelization* is a polynomial time (in the size of I and k) reduction $(I, k) \mapsto (I', k')$ such that I is a Yes-instance if and only if I' is a Yes-instance, the size of I' is bounded (from above) by a function $g(k)$ depending on k only and $k' \leq k$. The instances (I', k') comprise a *kernel* of Π of size $g(k)$.

Let us return to k -VC and consider the following kernelization rules. We start from the empty vertex cover C .

1. If G has an isolated vertex, remove it from G .

2. If G has a vertex x of degree 1, then add to C the neighbor y of x and delete y from G . Decrease k by 1.
3. If G has a vertex z of degree at least $k + 1$, add z to C and delete z from G . Decrease k by 1.

Rule 1 is obvious as no isolated vertex need to be in the vertex cover. Rule 2 is justified since a vertex of degree 1 in a vertex cover can be replaced by its neighbor and Rule 3 is justified since if a vertex z of degree at least $k + 1$ is not in a vertex cover, the vertex cover must have at least $k + 1$ vertices as it must contain all neighbors of z . Kernelization for k -VC, that uses the three rules as long as at least one of them is applicable, gives us a new graph G' and a new parameter k' equal to the current value of k . Since the degree of every vertex in G' is at most k , each vertex of G' can cover at most k edges of G' . Thus, for G to be a Yes-instance of k -VC, G' must have at most k^2 edges.

Research on kernelizations has already produced several important methods and approaches of kernelization of parameterized problems that allow one to produce small kernels for some problems (for k -VC there are reductions to kernels with at most $2k$ vertices). For more information, see the FPA monographs [13, 16, 42] and a survey paper [24].

Kernelizations are of interest when we know in advance that the parameter k is small relatively to the size of the problem instance and if the kernels are polynomial or moderately exponential function. Polynomial-size kernels are of special interest, but many FPT parameterized problem are very unlikely to have such kernels, see a recent paper [8], where the authors provide a widely applicable condition for an FPT parameterized problem not to have polynomial-size kernel (unless $\text{coNP} \subseteq \text{NP}/\text{poly}$, which is extremely unlikely).

3 Probabilistic Analysis of Testbeds

For the purpose of a heuristic evaluation it is important to know the optimal solution of a given instance. However, it is often impossible to find the optimal solutions of the instances used in computational experiments in a reasonable time. Instead of that one can use a probabilistic estimation of the average optimal solution for an instance family.

Probabilistic analysis of instances of various optimization problems has been carried out by several authors, see, e.g., the monographs [2, 9]. In this section we demonstrate some approaches for the average optimal solution estimation for two combinatorial problems: the Multidimensional Assignment Problem and the Traveling Salesman Problem.

3.1 Multidimensional Assignment Problem

The *Multidimensional Assignment Problem* (MAP) (abbreviated s -AP in the case of s dimensions) is an extension of a well-known *Assignment Problem* (AP) which is exactly the two dimensional case of MAP. While AP can be solved in a polynomial time [40], s -AP for every $s > 2$ is NP-hard [19]. For a fixed $s \geq 2$, the s -AP is stated as follows. Let $X_1 = X_2 = \dots = X_s = \{1, 2, \dots, n\}$. We will consider only vectors that belong to the Cartesian product $X = X_1 \times X_2 \times \dots \times X_s$. Each vector $e \in X$ is assigned a non-negative weight $w(e)$. For a vector $e \in X$, the component e_j denotes

its j th coordinate, i.e., $e_j \in X_j$. A collection A of $t \leq n$ vectors e^1, e^2, \dots, e^t is a (feasible) partial assignment if $e_j^i \neq e_j^k$ holds for each $i \neq k$ and $j \in \{1, 2, \dots, s\}$. The weight of a partial assignment A is $w(A) = \sum_{i=1}^t w(e^i)$. An assignment (or full assignment) is a partial assignment with n vectors. The objective of s -AP is to find an assignment of minimum weight.

We will sometimes use the notation x -assignment for an assignment of weight x .

One of the most used instance families for MAP is a *Random Instance Family* (see [29] and references therein). In Random instances, the weight assigned to a vector is an independent random uniformly distributed integer value in the interval $[a, b - 1]$.

It is possible to estimate the average solution value for the Random Instance Family. In fact, we prove that it is very likely that every large enough Random instance has an an -assignment, i.e., a minimal possible assignment (note that a minimal assignment includes n vectors of weight a).

Let α be the number of assignments of weight na and let $c = b - a$. We would like to have an upper bound on the probability $\Pr(\alpha = 0)$. Such an upper bound is given in the following theorem whose proof in [29] is based on the Extended Jansen Inequality given in Theorem 8.1.2 of [2].

Theorem 5. For values of n such that $n \geq 3$ and

$$\left(\frac{n-1}{e}\right)^{s-1} \geq c \cdot 2^{\frac{1}{n-1}}, \quad (1)$$

we have $\Pr(\alpha = 0) \leq e^{-\frac{1}{2\sigma}}$, where $\sigma = \sum_{k=1}^{n-2} \frac{\binom{n}{k} \cdot c^k}{[n \cdot (n-1) \cdots (n-k+1)]^{s-1}}$.

Proof. Let $[n] = \{1, 2, \dots, n\}$. Let t be the number of all feasible assignments and let A be an arbitrary assignment consisting of vectors e^1, e^2, \dots, e^n such that $e_1^i = i$ for each $i \in [n]$. There are $n!$ possibilities to choose the j th coordinate of all vectors in A for each $j = 2, 3, \dots, n$ and, thus, $t = (n!)^{s-1}$.

Let R be the set of vectors in X of weight a and let $\{A_1, A_2, \dots, A_t\}$ be the set of all assignments. Let B_i be the event $\{A_i \subset R\}$ for each $i \in [n]$. Let $\mu = \sum_{i=1}^t \Pr(B_i)$ and $\Delta = \sum_{i \sim j} \Pr(B_i \cap B_j)$, where $i \sim j$ if $i \neq j$ and $A_i \cap A_j \neq \emptyset$ and the sum for Δ is taken over all ordered pairs (B_i, B_j) with $i \sim j$.

By the Extended Jansen Inequality,

$$\Pr(\alpha = 0) \leq e^{-\frac{\mu^2}{2\Delta}} \quad (2)$$

provided $\Delta \geq \mu$. We will compute μ and estimate Δ to apply (2) and to show $\Delta \geq \mu$. It is easy to see that $\mu = \frac{t}{c^n}$.

Now we will estimate Δ . Let $A_i \cap A_j = K$, $k = |K|$ and $i \neq j$. Thus, we have

$$\Pr(B_i \cap B_j) = \Pr(K \subset R) \cdot \Pr(A_i \setminus K \subset R) \cdot \Pr(A_j \setminus K \subset R) = \frac{1}{c^k} \left(\frac{1}{c^{n-k}}\right)^2 = \frac{1}{c^{2n-k}}.$$

Let (f^1, f^2, \dots, f^n) be an assignment with $f_1^i = i$ for every $i \in [n]$ and consider the following two sets of assignments. Let

$$P(k) = \{(e^1, e^2, \dots, e^n) : \forall i \in [n] (e_1^i = i) \text{ and } \forall j \in [k] (e^j = f^j)\}$$

and let $Q(n-k) = \{(e^1, e^2, \dots, e^n) : \forall i \in [n] (e_i^i = i) \text{ and } \forall j \in [n-k] (e^{k+j} \neq f^{k+j})\}$. Let $h(n, k) = |P(k) \cap Q(n-k)|$. Clearly, $h(n, k) \leq |P(k)| = ((n-k)!)^{s-1}$. Observe that

$$h(n, k) \geq |P(k)| - (n-k)|P(k+1)| = L(n, k, s),$$

where $L(n, k, s) = ((n-k)!)^{s-1} - (n-k) \cdot ((n-k-1)!)^{s-1}$.

Let $g(n, k)$ be the number of ordered pairs (A_i, A_j) such that $|A_i \cap A_j| = k$. Observe that $g(n, k) = t \cdot \binom{n}{k} \cdot h(n, k)$ and, thus, $t \cdot \binom{n}{k} \cdot L(n, k, s) \leq g(n, k) \leq t \cdot \binom{n}{k} \cdot ((n-k)!)^{s-1}$.

Observe that $\Delta = \sum_{k=1}^{n-2} \sum_{|A_i \cap A_j|=k} \Pr(B_i \cap B_j) = \sum_{k=1}^{n-2} g(n, k) \cdot c^{k-2n}$. Thus,

$$\frac{(n!)^{s-1}}{c^{2n}} \cdot \sum_{k=1}^{n-2} \binom{n}{k} \cdot c^k \cdot L(n, k, s) \leq \Delta \leq \frac{(n!)^{s-1}}{c^{2n}} \sum_{k=1}^{n-2} \binom{n}{k} \cdot c^k \cdot ((n-k)!)^{s-1} \quad (3)$$

Now $\Pr(\alpha = 0) \leq e^{-\frac{1}{2s}}$ follows from (2) by substituting μ with $\frac{(n!)^{s-1}}{c^n}$ and Δ with its upper bound in (3). It remains to prove that $\Delta \geq \mu$. Since $n \geq 3$, $L(n, 1, s) \geq \frac{1}{2}((n-1)!)^{s-1}$. By the lower bound for Δ in (3), we have $\Delta \geq \frac{(n!)^{s-1}}{c^{2n-1}} \cdot L(n, 1, s)$. Therefore, $\frac{\Delta}{\mu} \geq \frac{0.5((n-1)!)^{s-1}}{c^{n-1}}$. Now using the inequality $(n-1)! > (\frac{n-1}{e})^{n-1}$, we conclude that $\frac{\Delta}{\mu} \geq 1$ provided (1) holds. \square

The proof is provided here since this technique can be used for obtaining similar results for some problems.

Useful results can also be obtained from (11) in [22] that is an upper bound for the average optimal solution. Grundel, Oliveira and Pardalos [22] consider the same instance family except the weights of the vectors are real numbers uniformly distributed in the interval $[a, b]$. However the results from [22] can be extended to our discrete case. Let $w'(e)$ be a real weight of the vector e in a continuous instance. Consider a discrete instance with $w(e) = \lfloor w'(e) \rfloor$ (if $w'(e) = b$, set $w(e) = b-1$). Note that the weight $w(e)$ is a uniformly distributed integer in the interval $[a, b-1]$. The optimal assignment weight of this instance is not larger than the optimal assignment weight of the continuous instance and, thus, the upper bound for the average optimal solution for the discrete case is correct.

In fact, the upper bound \bar{z}_u^* (see [22]) for the average optimal solution is not really accurate. For example, $\bar{z}_u^* \approx an + 6.9$ for $s = 3$, $n = 100$ and $b - a = 100$, and $\bar{z}_u^* \approx an + 3.6$ for $s = 3$, $n = 200$ and $b - a = 100$. It gives a better approximation for larger values of s , e.g., $\bar{z}_u^* \approx an + 1.0$ for $s = 4$, $n = 40$ and $b - a = 100$, but Theorem 5 provides stronger results ($\Pr(\alpha > 0) \approx 1.000$ in the latter case).

The following table taken from [29] gives the probabilities for $\Pr(\alpha > 0)$ for various values of s and n . The table was used in [29] to show that the optimal solutions of all the considered s -AP instances for $s \geq 4$ are almost surely to be the minimal possible solutions.

$s = 4$		$s = 5$		$s = 6$		$s = 7$	
n	$\Pr(\alpha > 0)$	n	$\Pr(\alpha > 0)$	n	$\Pr(\alpha > 0)$	n	$\Pr(\alpha > 0)$
15	0.575	10	0.991	8	1.000	7	1.000
20	0.823	11	0.998				
25	0.943	12	1.000				
30	0.986						
35	0.997						
40	1.000						

3.2 Travelling Salesman Problem

The *Asymmetric Traveling Salesman Problem (ATSP)* is the problem of computing a minimum weight tour (Hamilton directed cycle) passing through every vertex in a weighted complete digraph K_n^* on n vertices. The *Symmetric TSP (STSP)* is the same problem but on a weighted complete undirected graph K_n . When a certain fact holds for both ATSP and STSP, we will simply speak of *TSP*. We often assume that the vertices of K_n^* and K_n are $1, 2, \dots, n$ and refer to the weight $w(ij)$ of an edge of K_n^* (or K_n) as the *distance* from i to j .

A widely used special case of TSP is *Euclidean TSP*, i.e., TSP with the Euclidean distance. Random Euclidean TSP instances of large sizes are often used in literature and, thus, an estimation of the optimal solution of such instances is of our interest.

Let $V^n = \{v_1, v_2, \dots, v_n\}$ be a set of independent random variables uniformly distributed in $[0, 1]^d$, where $d \geq 2$. We define *Random Euclidean TSP Instance* as a TSP instance on a graph with a vertex set V and Euclidean distance function. Let $T(V^n)$ be the weight of a shortest Hamiltonian cycle through every vertex in V^n .

The following result has been first proved by Beardwood, Halton, and Hammersley [5]:

Theorem 6. *For all $d = 2, 3, \dots$ there exists a finite positive constant $\beta(d)$ such that*

$$\lim_{n \rightarrow \infty} \frac{T(V^n)}{n^{(d-1)/d}} = \beta(d)$$

with probability one.

It appears [21] that the value of $\beta(2)$ is approximately 0.7124 ± 0.0002 , so one can use the following estimation for the two-dimensional case: $T(V^n) \approx 0.7124\sqrt{n}$. However, the estimation of the expected solution convergences quite slow and it yields a better result just to add 0.7% to the Held-Karp lower bound [36]. According to [37], the expected value of the Held-Karp lower bound for the Random Euclidean Instance can be approximated with the following empirical formula:

$$\sqrt{n} \cdot (0.70805 + 0.52229n^{-0.5} + 1.31772n^{-1} - 3.07474n^{-1.5}).$$

4 Why Some Greedy Type Heuristics Fail

The aim of this section is to warn the reader that not always a greedy like approach is a good option and, in certain cases, it is a very bad option being sometimes among the worst possible options. Our message is not a discouragement from using greedy like algorithms altogether; we believe that for every combinatorial optimization problem

of importance, researchers and practitioners should simply investigate the appropriateness of greedy like algorithms and the existence of better alternatives to them (considering both quality of solution and running time). In many cases, especially when the running time must be very short, the conclusion may still be that the most practical of known approaches is a greedy like algorithm.

There are some general theoretical results that indicate that there are, in fact, many combinatorial optimization problems for which greedy like algorithms are not the best option even among fast construction heuristics, see, e.g., [3, 6, 32]. We will not consider these general results in order to avoid most mathematical details that are not necessary for understanding the results of this section, but we will show how they can be applied to two combinatorial problems, TSP and MAP, for which greedy like approaches are usually not very successful.

It is not a trivial question whether a certain algorithm is greedy like or not. Next we define an independence system and give the classic definition of the greedy algorithm for such a system. We extend this definition to so-called greedy type algorithms that include such well-known algorithms as the Prim's algorithm for the minimum spanning tree problem and the nearest neighbor algorithm for the traveling salesman problem. We use the term 'greedy like' in an informal way and we include in this class simple and fast construction heuristics that seem to be of greedy nature. Unfortunately, no formal definition exists for the wide family of greedy like algorithms and one can understand the difficulty to formally classify such algorithms by, for example, considering local search algorithms which find the best solution in each neighborhood they search. Intuitively, it is clear that such local search algorithms are not greedy yet their every search is greedy in a sense.

Let \mathcal{P} be a combinatorial optimization problem and let \mathcal{H} be a heuristic for \mathcal{P} . The *domination number* $\text{domn}(\mathcal{H}, \mathcal{I})$ of \mathcal{H} for an instance \mathcal{I} of \mathcal{P} is the number of solutions of \mathcal{I} that are not better than the solution s produced by \mathcal{H} including s itself. For example, consider an instance \mathcal{T} of the ATSP on 4 vertices depicted in Fig. 1. The weights of tours in \mathcal{T} are 13, 14, 17, 17, 20, 21. Suppose that the greedy algorithm computes a tour T of weight 17. Then $\text{domn}(\text{greedy}, \mathcal{T}) = 4$. In general, if $\text{domn}(\mathcal{H}, \mathcal{I})$ equals the number of solutions in \mathcal{I} , then \mathcal{H} finds an optimal solution for \mathcal{I} . If $\text{domn}(\mathcal{H}, \mathcal{I}) = 1$, then the solution found by \mathcal{H} for \mathcal{I} is the unique worst possible one. The *domination number* $\text{domn}(\mathcal{H}, n)$ of \mathcal{H} is the minimum of $\text{domn}(\mathcal{H}, \mathcal{I})$ over all instances \mathcal{I} of size n .

Notice that the domination number depends only on the ranking of solutions rather than the actual values of solutions.

For a formal definition of a greedy or a greedy type algorithms we should introduce the following. An *independence system* is a pair consisting of a finite set E and a family \mathcal{F} of subsets (called *independent sets*) of E such that (I1) and (I2) are satisfied:

- (I1) The empty set is in \mathcal{F} ;
- (I2) If $X \in \mathcal{F}$ and Y is a subset of X , then $Y \in \mathcal{F}$.

All maximal sets of \mathcal{F} are called *bases* (or, *feasible solutions*).

Many combinatorial optimization problems can be formulated as follows. We are given an independence system (E, \mathcal{F}) , a set $W \subseteq \mathbb{Z}_+$ and a weight function w that assigns a weight $w(e) \in W$ to every element of E (\mathbb{Z}_+ is the set of non-negative integers). The weight $w(S)$ of $S \in \mathcal{F}$ is defined as the sum of the weights of the elements of S . It is required to find a base $B \in \mathcal{F}$ of minimum weight. We will consider only such problems and call them the (E, \mathcal{F}, W) -*optimization problems*.

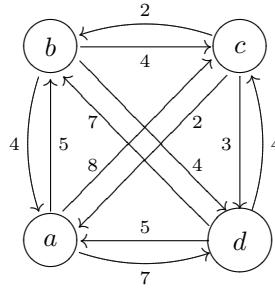


Fig. 1: An example of four vertex ATSP instance \mathcal{T} . There are six possible tours of the following weights in \mathcal{T} : 13, 14, 17, 17, 20, 21. Starting from a , **greedy** finds the tour $abcd$ of weight 17. Thus, $\text{domn}(\text{greedy}, \mathcal{T}) = 4$.

TSP is an $(E, \mathcal{F}, \mathbb{Z}_+)$ -optimization problem, where E is the set of arcs of the complete digraph K_n^* and $\mathcal{F} = \{B \subseteq H : H \in \mathcal{H}\}$, where \mathcal{H} is the set of Hamilton directed cycles of K_n^* . MAP is also an $(E, \mathcal{F}, \mathbb{Z}_+)$ -optimization problem, where E is the set of all vectors and \mathcal{F} is the set of all partial assignments.

If $S \in \mathcal{F}$, then let $I(S) = \{x : S \cup \{x\} \in \mathcal{F}\} \setminus S$. This means that $I(S)$ consists of those elements from $E \setminus S$, which can be added to S , in order to have an independent set of size $|S| + 1$. Note that by (I2) $I(S) \neq \emptyset$ for every independent set S which is not a base.

The *Greedy Algorithm* tries to construct a minimum weight base as follows: it starts from an empty set X , and at every step it takes the current set X and adds to it a minimum weight element $e \in I(X)$, the algorithm stops when a base is built. We assume that the greedy algorithm may choose any element among equally weighted elements in $I(X)$. Thus, when we say that the greedy algorithm *may construct* a base B , we mean that B is built provided the appropriate choices between elements of the same weight are made.

A *greedy type* algorithm \mathcal{H} , introduced in [30] is similar to the greedy algorithm: start with the partial solution $X = \emptyset$; and then repeatedly add to X an element of minimum weight in $I_{\mathcal{H}}(X)$ (ties are broken arbitrarily) until X is a base of \mathcal{F} , where $I_{\mathcal{H}}(X)$ is a subset of $I(X)$ that does not depend on the cost function c , but only on the independence system (E, \mathcal{F}) and the set X . Moreover, $I_{\mathcal{H}}(X)$ is non-empty if $I(X) \neq \emptyset$, a condition that guarantees that \mathcal{H} always outputs a base.

An example of a greedy type algorithm for TSP is the *Nearest Neighbor* algorithm: start from an arbitrary vertex i_1 and go to a vertex $i_2 \neq i_1$ with shortest distance from i_1 ; when in a vertex i_k , $k < n$, go to a vertex i_{k+1} with shortest distance from i_k among vertices not in the set $\{i_1, i_2, \dots, i_{k-1}\}$. An example of a greedy type algorithm for MAP is the *First Coordinate Fixing* algorithm: at the i th iteration, $i = 1, 2, \dots, n$, choose a vector e^i of minimum weight such that $e_1^i = i$ and $\{e^1, e^2, \dots, e^i\}$ is a partial assignment.

Theorem 7. *It is true for TSP with $n \geq 3$ and for MAP with $s \geq 2$ and $n \geq 1$ that for every greedy type algorithm there exists at least one instance such that the algorithm finds the unique worst possible solution.*

The proofs of this theorem for TSP [6] and for MAP [27] are not constructive and use the main result of [6].

Since greedy algorithm is always a greedy type algorithm (in the case $I_{\mathcal{H}}(X) = I(X)$), the domination number of any greedy or greedy type algorithm for TSP or MAP is 1.

Consider a greedy-like heuristic *max-regret-fc* first introduced by Ghosh et al. [20]. While the solution is not complete, choose i such that the value $w(ik) - w(ij)$ (called *regret*) is maximized, where the arcs ik and ij , $w(ik) \geq w(ij)$, are the lightest arcs among all the arcs from the vertex i that can be add to the current solution, and add the arc ij to the solution. A similar heuristic *max-regret* chooses the maximal regret not only among $w(ik) - w(ij)$ but also among $w(ki) - w(ji)$. It is true the domination numbers of both max-regret and max-regret-fc are 1. A very similar heuristics can be formulated for MAP, and the domination numbers of these heuristics still will be 1.

At this stage the reader may ask the following natural question: ‘Perhaps, it is true that every TSP or MAP heuristic has the domination number equal 1?’ The answer is negative. In fact, there are many TSP and MAP heuristics (see, e.g., [26, 29, 31, 44]) with larger domination numbers. Here is an incomplete list of TSP heuristics with exponential (at least $\Theta((n-2)!)$) domination number: several vertex insertion heuristics, k -Opt local search ($k \geq 2$ for STSP and $k \geq 3$ for ATSP) and Lin-Kernighan heuristic [31, 44]). These heuristics guarantee that their solution is not worse than the average solution. This property holds also for some MAP heuristics of exponential domination number such as 3-Opt local search (for 3-AP) and Recursive Opt Matching [26].

5 Very Large Neighborhood Local Search: Theory and Practice

Local Search (LS) is one of the most successful approaches in constructing heuristics for combinatorial optimization problems. Recently, several researchers investigated LS with Very Large Scale Neighborhoods (see, e.g., [1, 10, 12, 15, 33, 35, 43]). The authors of [1] point out that “as a rule of thumb, the larger the neighborhood, the better is the quality of the locally optimal solutions, and the greater is the accuracy of the final solution that is obtained.” However, several computational experiments do not support this rule, see, e.g., [10, 14, 35], where LS algorithms with small neighborhoods are superior to those with large neighborhoods. This means that it is not the size of a neighborhood, but some other parameters that are responsible for the relative power of the neighborhood. Gutin and Yeo [32] observed that theoretical and experimental results on TSP indicate that one such parameter may well be the domination number of the corresponding LS. Certainly, other parameters may also play a significant role. One such parameter is the diameter of the neighborhood digraph defined and discussed later on.

Very large domination number (see Section 4) and finite diameter of the neighborhood digraph are no guarantee that a local search algorithm with a Very Large Scale Neighborhood will be successful, so computational experiments are still necessary. However, we believe that a very small domination number (with respect to the total number of solutions) and/or infinite diameter of the neighborhood digraph indicate that the corresponding LS algorithm is likely to be of poor quality. Note that researchers have developed several high quality LS algorithms based on Very Large Scale Neighborhoods, see, e.g., [15, 18]. In particular, De Franceschi, Fischetti and Toth [18] extended a well-known polynomial-time searchable Punnen’s neighborhood for TSP (Punnen’s neighborhood [43] is an extension of the neighborhood `assign` described below) to a non-polynomial time searchable neighborhood. Then they used

an Integer Linear Programming solver to find a tour that is not necessarily best in its domain, but very close to the optimal.

In this section, to keep our discussion short, we consider only a few neighborhoods for the Asymmetric TSP (ATSP). The papers [1, 10, 12, 15, 29, 33, 35, 43] and many others consider many more neighborhoods for ATSP as well as for many other optimization problems. We adopt the definition of a neighborhood for the ATSP due to Deineko and Woeginger [12] (certainly, there are useful neighborhoods that are not captured by this definition, but the vast majority satisfy this definition). Let Π be a set of permutations on n vertices. Then the *neighborhood* with respect to Π of a tour $T = u_1 u_2 \dots u_n u_1$ is defined as follows:

$$N_{\Pi}(T) = \{u_{\pi(1)}u_{\pi(2)} \dots u_{\pi(n)}u_{\pi(1)} : \pi \in \Pi\}.$$

We are interested only in *polynomial-time searchable* neighborhoods, i.e., neighborhoods where the best solution can be found in polynomial time.

One of the first exponential size TSP neighborhoods (called **assign** in [12]) was considered independently by Sarvanov and Doroshko [46], and Gutin [25]. We describe this neighborhood and establish a simple upper bound on the domination number of the best improvement LS based on this neighborhood. We will see that the domination number of the best improvement LS based on **assign** is significantly smaller than that of the best improvement LS based on **3-opt**, a well-known ATSP heuristic. (A *best improvement* LS computes the best tour in every iteration.)

Consider an instance \mathcal{I} of ATSP on $n = 2k$ vertices and let $T = x_1 y_1 x_2 y_2 \dots x_k y_k x_1$ be an arbitrary tour of \mathcal{I} . The neighborhood **assign**, $N_a(T)$, of T is defined as follows: $N_a(T) = \{x_1 y_{\pi(1)} x_2 y_{\pi(2)} \dots x_k y_{\pi(k)} x_1 : (\pi(1), \pi(2), \dots, \pi(k)) \text{ is a permutation of } (1, 2, \dots, k)\}$. Clearly, $N_a(T)$ contains $k!$ tours. We will show that we can find the tour of minimum weight in $N_a(T)$ in polynomial time.

Let B be a complete bipartite graph with partite sets $\{z_1, \dots, z_n\}$ and $\{y_1, \dots, y_n\}$, and let the weight of $z_i y_j$ be $w(x_i y_j) + w(y_j x_{i+1})$ (where $x_{n+1} = x_1$). Let M be a perfect matching in B , and assume that z_i is matched to $y_{m(i)}$ in M . Observe that the weight of M is equal to the weight of the tour $x_1 y_{m(1)} x_2 y_{m(2)} \dots x_n y_{m(n)} x_1$. Since every tour in $N_a(T)$ corresponds to a perfect matching in B , and vice versa, a minimum weight perfect matching in B corresponds to a minimum weight tour in $N_a(T)$. Since we can find a minimum weight perfect matching in B in $O(n^3)$ time using the Hungarian method, we obtain the following theorem.

Theorem 8. [46, 25] *The best tour in $N_a(T)$ can be found in $O(n^3)$ time.*

While the size of $N_a(T)$ is quite large, the domination number of the best improvement LS based on **assign** is relatively small. Indeed, suppose that the weights of all arcs of the forms $x_i y_j$ and $y_j x_j$ equal 1 and the weights of all other arcs equal 0. Then, starting from the tour $T = x_1 y_1 x_2 y_2 \dots x_k y_k x_1$ of weight n the best improvement heuristic will output a tour of weight n , too. However, there are only $(k!)^2/k$ tours of weight n in \mathcal{I} and the weight of no tour in \mathcal{I} exceeds n . We have obtained the following:

Proposition 9. *For STSP, the domination number of the best improvement LS based on **assign** is at most $(k!)^2/k$, where $k = n/2$.*

The k -opt, $k \geq 3$, neighborhood of a tour T consists of all tours that can be obtained by deleting a collection of k arcs and adding another collection of k arcs. It is easy to see that one iteration of the best improvement k -opt LS can be completed

in time $O(n^k)$. Punnen, Margot and Kabadi [44] proved the following important result.

Theorem 10. *For the ATSP the best improvement 3-opt LS produces a tour, which is not worse than at least $(n - 2)!$ other tours, in at most $O(n^3 \log n)$ iterations.*

The last two assertions imply that after a polynomial number of iterations the best improvement 3-opt LS has domination number at least $\Omega(2^n/n^{1.5})$ times larger than that of the best improvement assign LS.

Computational experiments show clearly that 3-opt LS produces much better tours than LS based on assign does. Another theoretical explanation of this phenomena is based on the notion of the diameter of the neighborhood digraph. Let \mathcal{T} be the set of all tours of an ATSP instance and let N be an ATSP neighborhood. The neighborhood digraph D of N has vertex set \mathcal{T} and there is an arc from a tour T' to a tour $T'' \neq T'$ if $T'' \in N(T')$. The diameter of D is the maximum over all distances between vertices in D (the distances are measured in the number of arcs). If there is no path between some vertices of D then the diameter is ∞ .

It is not difficult to see that the diameter of the neighborhood digraph of N_a is ∞ . Indeed, let $T = x_1y_1x_2y_2 \dots x_ky_kx_1$ and $T' = x'_1y'_1x'_2y'_2 \dots x'_ky'_kx'_1$ be a pair of tours such that $\{x_1, \dots, x_k\} \cap \{x'_1, \dots, x'_k\} \neq \emptyset$ and $\{x_1, \dots, x_k\} \neq \{x'_1, \dots, x'_k\}$. Then there is no directed path from T to T' in the neighborhood digraph of N_a . For the diameter $\text{diam}_{3\text{-opt}}(n)$ of the 3-opt neighborhood digraph the situation is different.

Proposition 11. *For ATSP on $n \geq 3$ vertices, we have $\text{diam}_{3\text{-opt}}(n) \leq n - 2$.*

Proof. Consider a tour $T_0 = x_1x_2 \dots x_nx_1$, where $x_1 = 1$. At the i 'th iteration ($i \geq 1$), we will get a tour $T_i \in N_3(T_{i-1})$, where N_3 denotes the 3-opt neighborhood. It is sufficient to prove that we can choose T_1, T_2, \dots, T_p with $p \leq n - 2$ such that $T_p = 12 \dots n1$. Our proof is by induction on $n \geq 3$.

The basis case $n = 3$ is trivial. Consider the $(n - 1)$ -vertex ATSP where x_1 and x_2 are contracted to one vertex x'_1 . Now instead of T_0 we get $T'_0 = x_1x_3x_4 \dots x_nx_1$. Since we have $n - 1$ vertices rather than n vertices, by induction hypothesis, there is a sequence T'_1, T'_2, \dots, T'_q ($q \leq n - 3$) of tours such that $T'_i \in N_3(T'_{i-1})$ and $T'_q = x'_123 \dots (x_2 - 1)(x_2 + 1) \dots nx'_1$. Now replacing x'_1 by x_1x_2 , we get the sequence T_1, T_2, \dots, T_q of tours of the original ATSP, where $T_q = 1x_223 \dots (x_2 - 1)(x_2 + 1) \dots n1$ (recall that $x_1 = 1$). (If $x_2 = 2$, $T_q = 12 \dots n1$ and we are done, so we assume that $x_2 \neq 2$.) Delete the arcs $(1, x_2)$, $(x_2, 2)$ and $(x_2 - 1, x_2 + 1)$ from T_q and add to it the arcs $(1, 2)$, $(x_2 - 1, x_2)$ and $(x_2, x_2 + 1)$. Since the obtained tour is $12 \dots n1$ and $q + 1 \leq n - 2$, we are done. \square

The diameters of several TSP neighborhood digraphs were computed in [33].

6 Conclusion

We have discussed several theoretical tools useful in optimization heuristics design and analysis. Preprocessing is aimed at significantly decreasing the running time of a heuristic and we provided certain examples when preprocessing is very successful. Moreover, we saw that sometimes preprocessing plays the central role in the algorithm under consideration. We showed that probabilistic analysis of testbeds allows, in some cases, to find an exact solution making the experimental evaluation more meaningful. We saw that domination analysis is aimed at studying the worst case behavior of heuristics, but it may help in explaining experimental evaluation results

and it may yield some new successful heuristics. Finally, we discussed which theoretical parameters are of interest in choosing appropriate neighborhoods for local search heuristics.

In this paper, we have covered just a few fields and certainly there are other theoretical areas that are useful in design and analysis of heuristics. However, we believe that we have achieved the main goal of this paper by providing several good examples of linking experimental and theoretical approaches in heuristics design and analysis.

Acknowledgment. We are thankful to the referees for a number of useful remarks and suggestions.

References

- [1] R.K. Ahuja, Ö. Ergun, J.B. Orlin, and A.P. Punnen. A survey of very large-scale neighborhood search techniques. *Discrete Appl. Math.*, 123:75–102, 2002.
- [2] N. Alon and J. Spencer. *The Probabilistic Method*. John Wiley, second edition, 2000.
- [3] J. Bang-Jensen, G. Gutin, and A. Yeo. When the greedy algorithm fails. *Discrete Optimization*, 1:121–127, 2004.
- [4] J.J. Bartholdi. A good submatrix is hard to find. *Oper. Research Letters*, 1:190–193, 1982.
- [5] J. Beardwood, J.H. Halton, and J.M. Hammersley. The shortest path through many points. *Proc. Camb. Philos. Society*, 55:299–327, 1959.
- [6] G. Bendall and F. Margot. Greedy type resistance of combinatorial problems. *Discrete Optimization*, 3:288–298, 2006.
- [7] R.E. Bixby. Solving real-world linear programs: A decade and more of progress. *Operations Research*, 50:3–15, 2002.
- [8] H.L. Bodlaender, R.G. Downey, M.R. Fellows, and D. Hermelin. On problems without polynomial kernels. Technical Report UU-CS-2007-046, Utrecht University, The Netherlands, 2007.
- [9] B. Bollobas. *Random Graphs*. Cambridge University Press, 2001.
- [10] T. Brueggemann and J. Hurink. Two very large-scale neighborhoods for single machine scheduling. *OR Spectrum*, 29:513–533, 2007.
- [11] J. Chen, I.A. Kanj, and G. Xia. Simplicity is beauty: Improved upper bounds for vertex cover. Technical Report TR05-008, DePaul University, Chicago IL, 2005.
- [12] V.G. Deineko and G.J. Woeginger. A study of exponential neighbourhoods for the traveling salesman problem and the quadratic assignment problem. *Math. Program., Ser. A*, 87:519–542, 2000.
- [13] R.G. Downey and M.R. Fellows. *Parameterized Complexity*. Springer-Verlag, New York, 1999.
- [14] Moshe Dror and Larry Levy. A vehicle routing improvement algorithm comparison of a “greedy” and a matching implementation for inventory routing. *Comput. Oper. Res.*, 13(1):33–45, 1986.
- [15] Ö. Ergun, J.B. Orlin, and A. Steele-Feldman. Creating very large scale neighborhoods out of smaller ones by compounding moves. *Journal of Heuristics*, 12(1-2):115–140, 2006.
- [16] J. Flum and M. Grohe. *Parameterized Complexity Theory*. Springer-Verlag, 2006.

-
- [17] R. Fourer and D.M. Gay. Experience with a primal presolve algorithm. In *Large Scale Optimization: State of the art*, pages 135–154. Kluwer Academic Publishers, 1994.
- [18] R. De Franceschi, M. Fischetti, and P. Toth. A new ilp-based refinement heuristic for vehicle routing problems. *Math. Programming*, 105(2-3):471–499, 2006.
- [19] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness (Series of Books in the Mathematical Sciences)*. W. H. Freeman, January 1979.
- [20] D. Ghosh, B. Goldengorin, G. Gutin, and G. Jäger. Tolerance-based greedy algorithms for the traveling salesman problem. *Communications in DQM*, 10:52–70, 2007.
- [21] M. Grigni, E. Koutsoupias, and C. Papadimitriou. An approximation scheme for planar graph tsp. In *Proceedings of the 36th Annual Symposium on Foundations of Computer Science (FOCS'95)*, pages 640–645, Washington, DC, USA, 1995. IEEE Computer Society.
- [22] D. Grundel, C. Oliveira, and P. Pardalos. Asymptotic properties of random multidimensional assignment problems. *Journal of Optimization Theory and Applications*, 122(3):33–46, 2004.
- [23] N. Gulpinar, G. Gutin, G. Mitra, and A. Zverovitch. Extracting pure network submatrices in linear programs using signed graphs. *Discrete Applied Mathematics*, 137:359–372, 2004.
- [24] J. Guo and R. Niedermeier. Invitation to data reduction and problem kernelization. *SIGACT News*, 38(1):31–45, 2007.
- [25] G. Gutin. On an approach to solving the traveling salesman problem. In *Proceedings of the USSR Conference on System Research*, pages 184–185. Nauka, Moscow, 1984. (in Russian).
- [26] G. Gutin, B. Goldengorin, and J. Huang. Worst case analysis of max-regret, greedy and other heuristics for multidimensional assignment and traveling salesman problems. *Journal of Heuristics*, 14(2):169–181, 2008.
- [27] G. Gutin and D. Karapetyan. *Advances in Greedy Algorithms*, chapter Greedy Like Algorithms for the Traveling Salesman and Multidimensional Assignment Problems. 2008.
- [28] G. Gutin and D. Karapetyan. Generalized traveling salesman problem reduction algorithms. *Preprint in arXiv*, <http://arxiv.org/abs/0804.0735>, 2008.
- [29] G. Gutin and D. Karapetyan. Local search heuristics for the multidimensional assignment problem. In *Proc. of Graph Theory, Computational Intelligence and Thought*, Israel, 2008. To appear in Lect. Notes Comput. Sci.
- [30] G. Gutin, A. Vainshtein, and A. Yeo. When greedy-type algorithms fail. *Unpublished manuscript*, 2002.
- [31] G. Gutin and A. Yeo. Polynomial approximation algorithms for the TSP and the QAP with a factorial domination number. *Discrete Appl. Math.*, 119:107–116, 2002.
- [32] G. Gutin and A. Yeo. *Graph Theory, Combinatorics and Algorithms: Interdisciplinary Applications*, chapter Domination Analysis of Combinatorial Optimization Algorithms and Problems. Springer-Verlag, 2005.
- [33] G. Gutin, A. Yeo, and A. Zverovitch. *The Traveling Salesman Problem and its Variations (G. Gutin and A.P. Punnen, eds.)*, chapter Exponential Neighborhoods and Domination Analysis for the TSP, pages 445–487. Kluwer, Dordrecht, 2002.
- [34] P.-O. Gutman and I. Ioslovich. On the generalized wolf problem: Preprocessing of nonnegative large-scale linear programming problems with group constraints. *Autom. Remote Control*, 68(8):1401–1409, 2007.

-
- [35] J. Hurink. An exponential neighborhood for a one-machine batching problem. *OR Spectrum*, 21:461–476, 1999.
- [36] D.S. Johnson and L.A. McGeoch. *Local Search in Combinatorial Optimization*, chapter The Traveling Salesman Problem: A Case Study in Local Optimization, pages 215–310. Wiley & Sons, 1995.
- [37] D.S. Johnson, L.A. McGeoch, and E. E. Rothberg. Asymptotic experimental analysis for the held-karp traveling salesman bound. In *SODA '96: Proceedings of the seventh annual ACM-SIAM symposium on Discrete algorithms*, pages 341–350, Philadelphia, PA, USA, 1996. Society for Industrial and Applied Mathematics.
- [38] N. Krasnogor and J. Smith. Emergence of profitable search strategies based on a simple inheritance mechanism. In *International Genetic and Evolutionary Computation Conference (GECCO2001)*, pages 432–439, San Francisco, CA, 2001. Morgan Kaufmann Publishers.
- [39] N. Krasnogor and J.E. Smith. A tutorial for competent memetic algorithms: model, taxonomy and design issues. *IEEE Transactions on Evolutionary Computation*, 9(5):474–488, 2005.
- [40] H.W. Kuhn. The hungarian method for the assignment problem. *Naval Research Logistic Quarterly*, 2:83–97, 1955.
- [41] G. Laporte, H. Mercure, and Y. Nobert. A branch and bound algorithm for a class of asymmetrical vehicle routeing problems. *The Journal of the Operational Research Society*, 43:469–481, 1992.
- [42] R. Niedermeier. *Invitation to Fixed Parameter Algorithms*. Oxford University Press, Oxford, UK, 2006.
- [43] A.P. Punnen. The traveling salesman problem: new polynomial approximation algorithms and domination analysis. *Inform. Optim. Sci.*, 22:191–206, 2001.
- [44] A.P. Punnen, F. Margot, and S.N. Kabadi. TSP heuristics: domination analysis and complexity. *Algorithmica*, 35:111–127, 2003.
- [45] F. Samanlioglu, Jr. W. G. Ferrell, and M. E. Kurz. A memetic random-key genetic algorithm for a symmetric multi-objective traveling salesman problem. *Comput. Ind. Eng.*, 55(2):439–449, 2008.
- [46] V.I. Sarvanov and N.N. Doroshko. The approximate solution of the traveling salesman problem by a local algorithm with scanning neighborhoods of factorial cardinality in cubic time. In *Software: Algorithms and Programs*, volume 31, pages 11–13. Math. Institute of the Belorussian Acad. Sci., Minsk, 1981. (in Russian).
- [47] M.W.P. Savelsbergh. Preprocessing and probing techniques for mixed integer programming problems. *ORSA Journal on Computing*, 6:445–454, 1994.