Complexity and Algorithms for the Traveling Salesman Problem and the Assignment Problem of Second Order*

Frank Fischer†, Gerold Jäger‡, Anja Lau†, Paul Molitor§

October 14, 2009

Abstract

We introduce two new combinatorial optimization problems, which are generalizations of the Traveling Salesman Problem (TSP) and the Assignment Problem (AP) and which we call Traveling Salesman Problem of Second Order (TSP2) and Assignment Problem of Second Order (AP2). TSP2 is motivated by an important application in bioinformatics, especially the Permuted Variable Length Markov model. While TSP2 is trivially $\mathcal{NP}$-hard, we show the $\mathcal{NP}$-hardness of AP2 by a reduction from SAT. We propose seven elementary heuristics for the TSP2, some of which are generalizations of similar algorithms for the Traveling Salesman Problem, some of which are new ideas. Furthermore we give four exact algorithms for the TSP2, namely a Branch-and-Bound (BnB) algorithm, an Integer Programming (IP) algorithm, a Branch-and-Cut (BnC) algorithm and an algorithm based on a polynomial reduction to the original TSP (TSP-R). Finally we experimentally compare the algorithms for many different random instances and real instances from the already mentioned application in bioinformatics. Our experiments show that for real instances most heuristics lead to optimal or almost-optimal solutions. For both, random and real classes, our most sophisticated exact approach BnC is the leading algorithm. In particular, the BnC algorithm is able to solve real instances up to size 80 in reasonable time, proving the applicability of this approach.

Keywords: Traveling Salesman Problem, Assignment Problem, Traveling Salesman Problem of Second Order, Assignment Problem of Second Order, Heuristic, Exact Algorithm.

MSC 2000: 90C27, 90C57

*A preliminary version of this paper appeared in the proceedings of the Second International Conference on Combinatorial Optimization and Applications (COCOA) 2008, Lecture Notes in Comput. Sci. 5165, p. 211-224.

†Fakultät für Mathematik, Technische Universität Chemnitz, D-09107 Chemnitz, Germany. {frank.fischer,anja.lau}@mathematik.tu-chemnitz.de

‡Institut für Informatik, Universität Kiel, D-24098 Kiel, Germany. gej@informatik.uni-kiel.de

§Institut für Informatik, Universität Halle-Wittenberg, D-06120 Halle (Saale), Germany. paul.molitor@informatik.uni-halle.de

1 Introduction

Gene regulation in higher organisms is accomplished by several cellular processes, one of which is transcription initiation. In order to better understand this process, it would be desirable to have a good understanding of how transcription factors bind to their binding sites. While tremendous progress has been made on the fields of structural biology and bioinformatics, the accuracies of existing models to predict the location and affinity of transcription factor binding sites are not yet satisfactory. The aim is to better understand gene regulation by finding more realistic binding site models. One model that extends the position weight matrix (PWM) model, the weight array matrix (WAM) model, and higher-order Markov models in a natural way is the Permuted Markov (PM) model. Permuted Markov models were proposed by [8] for the recognition of transcription factor binding sites. The class of PM models was further extended by [30] to the class of Permuted Variable Length Markov (PVLM) models, and it was demonstrated that PVLM models can improve the recognition of transcription factor binding sites for many transcription factors. Finding the optimal PM model for a given data set is $\mathcal{NP}$-hard, and finding the optimal PVLM model for a given data set is $\mathcal{NP}$-hard, too. Hence, heuristic algorithms for finding the optimal PM model and the optimal PVLM model were proposed in [8] and [30], respectively. Experimental evidence has been accumulated that suggests that the binding sites of many transcription factors fall into distinct classes. It has been proposed to extend PM models to PM mixture models, to extend PVLM models to PVLM mixture models, and to apply both mixture models to the recognition of transcription factor binding sites [17]. While both the PM mixture model and the PVLM mixture model look appealing from a biological perspective, they pose a computational challenge: the optimal PM mixture model and the optimal PVLM mixture model can be obtained only numerically. One of the commonly used algorithms for finding optimal mixture models is the Expectation Maximization (EM) algorithm. The EM algorithm consists of two steps, the E step and the M step, which are iterated until convergence. Applied to the problem of finding the optimal PM or PVLM mixture model of *order 1*, each M step requires the solution of a TSP instance. Likewise, applied to the problem of finding the optimal PM or PVLM mixture model of *order 2*, each M step requires the solution of an instance of a generalization of the TSP, which we call TSP2 and which is introduced in the following.

For a weighted directed graph the *Traveling Salesman Problem* (TSP) is the problem of finding a complete tour with minimal costs. TSP is $\mathcal{NP}$-hard, which can be shown by a simple polynomial reduction from the *Hamiltonian Cycle Problem* (HCP) and a polynomial reduction of HCP from the $\mathcal{NP}$-complete 3-SAT [24]. A related problem is the *Assignment Problem* (AP), which is to find a set of cycles with minimal costs, where each vertex is visited exactly once. There are many efficient algorithms for the AP [5, 12, 23] (for an experimental comparison of AP algorithms see [7]). Many algorithms are variations of the Hungarian method, which is based on König-Egervary's theorem and has a complexity of $\mathcal{O}(n^3)$ [26].

In this paper we introduce the following natural generalizations of TSP and AP, where the costs do not depend on arcs, but on each sequence of three consecutive vertices in

the tour and in the cycles, respectively. We call the corresponding problems *Traveling Salesman Problem of Second Order* (TSP2) and *Assignment Problem of Second Order* (AP2). To the best of our knowledge, these problems have not been considered in literature before. Although the number of feasible tours is $(n-1)!$ for both, TSP and TSP2, the latter is more difficult, as it uses a three-dimensional cost function and the TSP only a two-dimensional one. The same holds for the relation between AP and AP2 which both have $n!$ feasible solutions. The purpose of this paper is to show the complexity of both problems and to develop heuristics and exact algorithms and do an experimental study of these algorithms for random and real instances. As only TSP2 has applications in practice, we only consider algorithms for *this* problem, but we use AP2 algorithms as subroutines for TSP2 algorithms.

The paper is organized as follows. TSP2 and AP2 are defined in Section 2 and are shown to be $\mathcal{NP}$-hard in Section 3. Section 4 presents a polynomial reduction from TSP2 to TSP, which theoretically allows the application of TSP algorithms to TSP2. In Section 5 and 6 we propose different heuristics and exact algorithms, respectively, for the TSP2. In Section 7 we give a customization for symmetric instances and in Section 8 an experimental study for the heuristics and exact algorithms. Finally we summarize this paper and give suggestions for future research in Section 9.

2 Notations and Definitions

In the following let $G = (V, E)$ be a directed graph, where V is the set of vertices with $|V| = n$ and E the set of arcs.

TSP

Let $n \geq 2$ and a cost function $c : V \times V \to \mathbb{R} \cup \{\infty\}$ with $c(u, u) = \infty$ for all $u \in V$ be given. Then TSP is the problem of finding a complete tour $(v_1, v_2, \ldots, v_n)$ with minimal costs $c(v_n, v_1) + \sum_{j=1}^{n-1} c(v_j, v_{j+1})$. The special case that the costs of each arc equal the costs of the corresponding reverse arc, i.e., $c(v_i, v_j) = c(v_j, v_i)$ for all $1 \leq i \neq j \leq n$, is called *Symmetric Traveling Salesman Problem* (STSP). If this condition does not necessarily hold, the problem is also called *Asymmetric Traveling Salesman Problem* (ATSP).

AP

Let a cost function $c : V \times V \to \mathbb{R} \cup \{\infty\}$ be given. Then AP is the problem of finding a vertex permutation π with minimal costs $\sum_{i=1}^{n} c(v_i, v_{\pi(i)})$ over all possible permutations of $\{1, \ldots, n\}$. Note that each feasible AP solution is a set of cycles visiting each vertex exactly once. For our purpose, we additionally require $n \geq 2$ and $c(u, u) = \infty$ for $u \in V$.

TSP2

Let $n \geq 3$ and a cost function $c : V \times V \times V \to \mathbb{R} \cup \{\infty\}$ with $c(u, v, w) = \infty$ for $u, v, w \in V$ with $u = v$ or $u = w$ or $v = w$ be given. Then TSP2 is the problem of finding a complete tour $(v_1, v_2, \ldots, v_n)$ with minimal costs $c(v_{n-1}, v_n, v_1) + c(v_n, v_1, v_2) + \sum_{j=1}^{n-2} c(v_j, v_{j+1}, v_{j+2})$. Analogously to the TSP, consider the special case that the costs of each sequence of three consecutive vertices equal the costs of the corresponding reverse sequence of three consecutive vertices, i.e., $c(u, v, w) = c(w, v, u)$ for $u, v, w \in V$. We call this problem *Symmetric Traveling Salesman Problem of Second Order* (STSP2) and *Asymmetric Traveling Salesman Problem of Second Order* (ATSP2), otherwise.

AP2

Let a cost function $c : V \times V \times V \to \mathbb{R} \cup \{\infty\}$ be given. Then AP2 is the problem of finding a vertex permutation π with minimal costs $\sum_{i=1}^{n} c(v_i, v_{\pi(i)}, v_{\pi(\pi(i))})$ over all possible permutations of $\{1, \ldots, n\}$. For our purpose, we additionally require $n \geq 3$ and $c(u, v, w) = \infty$ for $u, v, w \in V$ with $u = v$ or $u = w$ or $v = w$.

3 TSP2 and AP2 are $\mathcal{NP}$-hard

As TSP2 is a generalization of the $\mathcal{NP}$-hard TSP, the $\mathcal{NP}$-hardness of TSP2 is easy to show.

Theorem 3.1. *TSP2 is $\mathcal{NP}$-hard.*

Proof. Let $c : V \times V \to \mathbb{R} \cup \{\infty\}$ be the cost function of a TSP instance. Now define the three-dimensional cost function $c' : V \times V \times V \to \mathbb{R} \cup \{\infty\}$ by $c'(u, v, w) := c(v, w) \ \ \forall u \in V \setminus \{v, w\}$. Then an optimal tour of the TSP instance is an optimal tour of the TSP2 instance, and vice versa. Thus TSP can be reduced in polynomial time to TSP2. □

The $\mathcal{NP}$-hardness of AP2 easily follows from the $\mathcal{NP}$-hardness of the more special Angular-Metric Traveling Salesman Problem [2] which is the problem of minimizing the total angle of a tour for a set of points in the Euclidian space, where the angle of a tour is the sum of the direction changes at the points. We show the $\mathcal{NP}$-hardness of AP2 in a much more elegant way by a reduction from SAT:

Theorem 3.2. *AP2 is $\mathcal{NP}$-hard.*

Proof. Let $\{C_1, \ldots, C_m\}$ be a set of clauses defined over the variables $\{x_1, \ldots, x_n\}$ and let us consider the SAT instance given by the conjunction of these clauses. Now we construct an AP2 instance, which is a directed graph $G = (V, E)$ with three-dimensional cost function $c : V \times V \times V \to \mathbb{R} \cup \{\infty\}$. The set of vertices V is defined as follows:

1. $v_{i,j} \in V$ for all $i \in N := \{1, \ldots, n\}, j \in \{0, \ldots, m\}$,

2. $u_j \in V$ for all $j \in M := \{1, \ldots, m\}$.

Thus $m + 1$ vertices exist for each variable, and there is one additional vertex for each clause. The set of arcs is comprised of several parts. For each variable $x_i, i \in N$, we have the following sets:

- E_i with the arcs
  - $(v_{i,m}, v_{i,0})$,
  - $(v_{i,j-1}, v_{i,j})$ for all $j \in M$,
  - $(v_{i,j-1}, u_j)$ and $(u_j, v_{i,j})$, if the literal x_i appears in clause C_j.
- $E_i^{\neg}$ with the arcs
  - $(v_{i,0}, v_{i,m})$,
  - $(v_{i,j}, v_{i,j-1})$ for all $j \in M$,
  - $(v_{i,j}, u_j)$ and $(u_j, v_{i,j-1})$, if the literal $\neg x_i$ appears in clause C_j.

The set of all arcs is $E = \bigcup_{i \in N}(E_i \cup E_i^{\neg})$. This means there exist two directed cycles $v_{i,0}, v_{i,1} \ldots v_{i,m}, v_{i,0}$ and $v_{i,m}, v_{i,m-1} \ldots v_{i,0}, v_{i,m}$ for each variable $x_i, i \in N$, where the first cycle can be extended by replacing the arc $(v_{i,j-1}, v_{i,j})$ by the two arcs $(v_{i,j-1}, u_j)$ and $(u_j, v_{i,j})$, if the literal x_i appears in clause C_j, and the second cycle can be extended by replacing the arc $(v_{i,j}, v_{i,j-1})$ by the two arcs $(v_{i,j}, u_j)$ and $(u_j, v_{i,j-1})$, if the literal $\neg x_i$ appears in clause C_j. Obviously, the construction above is possible in polynomial time. As an example, Fig. 1 shows the graph for the SAT instance $(x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg\, x_2 \vee \neg x_3) \wedge (x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_2)$.

The goal of AP2 is to cover all vertices with cycles of minimal costs. The arcs of E_i (the bold arcs in Fig. 1) correspond to setting the variable x_i to *TRUE*, and the arcs of $E_i^{\neg}$ correspond to setting the variable x_i to *FALSE*. We have to ensure that an AP2 solution uses only arcs from exactly one of those two sets for each variable x_i. Therefore, we use the following cost function. Let $(a, b), (b, c) \in E$ be two arcs. Then we set

$$c(a,b,c) = \begin{cases} 0, & \text{if } \exists i \in N : (a,b),(b,c) \in E_i \vee (a,b),(b,c) \in E_i^{\neg} \\ 1, & \text{otherwise} \end{cases}$$

In the following we show that a SAT instance is satisfiable, if and only if the corresponding AP2 solution has costs 0.

"$\Rightarrow$" Let $f : N \to \{TRUE, FALSE\}$ be a satisfying assignment of the variables, i.e., setting $x_i = f(i)$ satisfies all clauses. Thus for each C_j there is (at least) one literal $x_{h(j)}$ or $\neg x_{h(j)}$ contained in C_j which is *TRUE* ($h(j)$ denotes the index of one of those variables making C_j *TRUE*). We construct one cycle with costs 0 for each variable. Let x_i be an arbitrary variable. We distinguish the following two cases.

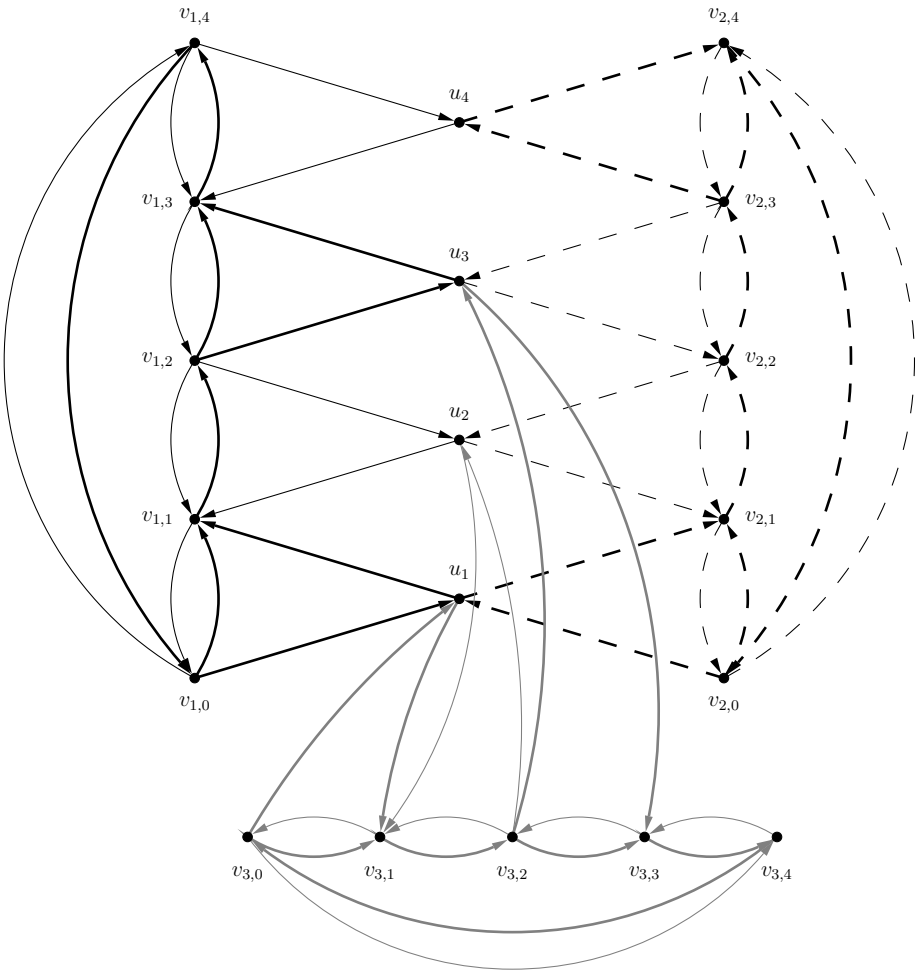


Figure 1: The graph for the example $(x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg\, x_2 \vee \neg x_3) \wedge (x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_2)$.

- $f(i) = \mathit{TRUE}$:

  If $h(j) = i$ for the clause C_j, use $(v_{i,j-1}, u_j), (u_j, v_{i,j}) \in E_i$ (the mentioned cycle extension), otherwise use $(v_{i,j-1}, v_{i,j}) \in E_i$. Finally use $(v_{i,m}, v_{i,0}) \in E_i$ to close the cycle.

- $f(i) = \mathit{FALSE}$:

  If $h(j) = i$ for the clause C_j, use $(v_{i,j}, u_j), (u_j, v_{i,j-1}) \in E_i^{\neg}$, otherwise use $(v_{i,j}, v_{i,j-1}) \in E_i^{\neg}$. Finally use $(v_{i,0}, v_{i,m}) \in E_i^{\neg}$ to close the cycle.

Those arcs form a cycle for each $i \in N$, cover all vertices and have costs 0.

"$\Leftarrow$" Let $W_1, \ldots, W_p$ be the cycles of an AP2 solution with costs 0. For $i = 1, \ldots, p$ define $V(W_i)$ as the set of vertices and $E(W_i)$ as the set of arcs of cycle W_i. By construction of the graph and the cost-function, each cycle uses arcs from exactly one set E_i or $E_i^{\neg}$ for $i \in N$. It holds $p = n$ and for each $i \in N$, either $E(W_i) \subset E_i$ or $E(W_i) \subset E_i^{\neg}$. We set x_i to *TRUE*, if $E(W_i) \subset E_i$, and x_i to *FALSE*, if $E(W_i) \subset E_i^{\neg}$. Since the cycles cover all vertices, for each $u_j, j \in M$, a cycle $W_{h(j)}$ exists such that $u_j \in V(W_{h(j)})$. Let $C_j, j \in M$, be an arbitrary clause. If $E(W_{h(j)}) \subset E_{h(j)}$, the variable $x_{h(j)}$ is set to *TRUE*, where the corresponding literal $x_{h(j)}$ appears in clause C_j by the construction of the graph. Analogously, if $E(W_{h(j)}) \subset E_{h(j)}^{\neg}$, the literal $x_{h(j)}$ is set to *FALSE*, where the corresponding literal $\neg x_{h(j)}$ appears in clause C_j. Consequently, this assignment fulfills all clauses.

For the considered example $(x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3) \wedge (x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_2)$, three SAT solutions exist, namely $x_1 = \mathit{TRUE}, x_2 = \mathit{TRUE}, x_3 = \mathit{FALSE}$; $x_1 = \mathit{FALSE}, x_2 = \mathit{TRUE}, x_3 = \mathit{TRUE}$; and $x_1 = \mathit{FALSE}, x_2 = \mathit{FALSE}, x_3 = \mathit{TRUE}$. Fig. 3 shows the two AP2 solutions for the first of these SAT solutions, where the arcs of $x_1 = \mathit{TRUE}$ are represented by solid lines, $x_2 = \mathit{TRUE}$ by dotted lines and $x_3 = \mathit{FALSE}$ by pale lines.

□

4 Polynomial Reduction from TSP2 to TSP

In the following we present a special polynomial reduction from TSP2 to TSP (more precisely to STSP).

Let a TSP2 instance $I_{\mathrm{TSP2}} = (V_{\mathrm{TSP2}}, c_{\mathrm{TSP2}})$ be given with vertex set V_{TSP2}, $|V_{\mathrm{TSP2}}| = n$, and cost function $c_{\mathrm{TSP2}} : V_{\mathrm{TSP2}} \times V_{\mathrm{TSP2}} \times V_{\mathrm{TSP2}} \to \mathbb{R}$. V_{TSP2} can be represented by the set $N := \{1, \ldots, n\}$. We construct a symmetric TSP, i.e., a STSP instance $I_{\mathrm{TSP}} = (V_{\mathrm{TSP}}, c_{\mathrm{TSP}})$ with vertex set $V_{\mathrm{TSP}} = \{v_{i,j,p} \,|\, i, j \in N,\ p \in \{0,1,2\}\}$ and cost function $c_{\mathrm{TSP}} : V_{\mathrm{TSP}} \times V_{\mathrm{TSP}} \to \mathbb{R}$. Define

$$K \ := \ 1 + n \cdot \max\{c_{\mathrm{TSP2}}(i,j,k) \,|\, i,j,k \in N,\ i \neq j, i \neq k, j \neq k\}.$$

For $i \in N$ define

$$\begin{aligned} C_i \ := \ & \{\{v_{i,j,p}, v_{i,j,p+1}\} \,|\, j \in N, p \in \{0,1\}\} \ \cup \ \{\{v_{i,j,2}, v_{i,j+1,0}\} \,|\, j \in N \setminus \{n\}\} \\ & \cup \{\{v_{i,n,2}, v_{i,1,0}\}\}. \end{aligned}$$

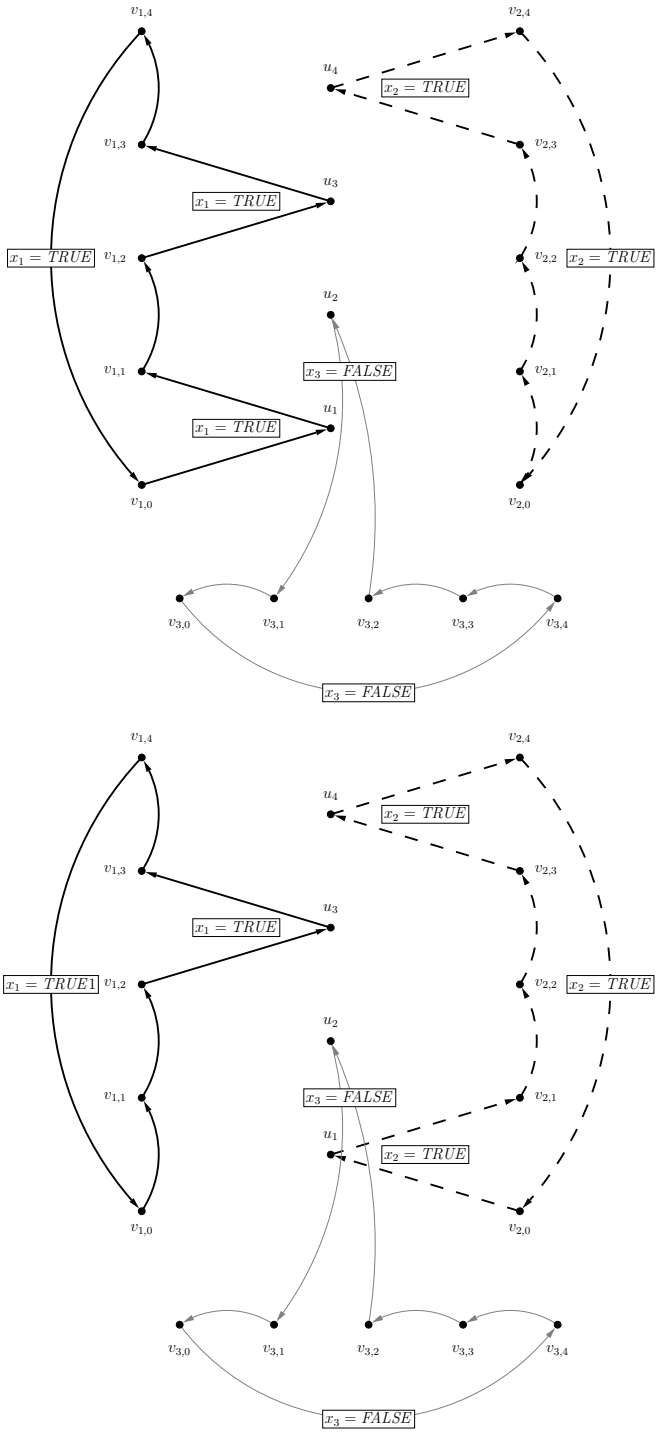


Figure 2: Two solutions for the instance $(x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg\, x_2 \vee \neg x_3) \wedge (x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_2)$ with $x_1 = TRUE, x_2 = TRUE, x_3 = FALSE$.

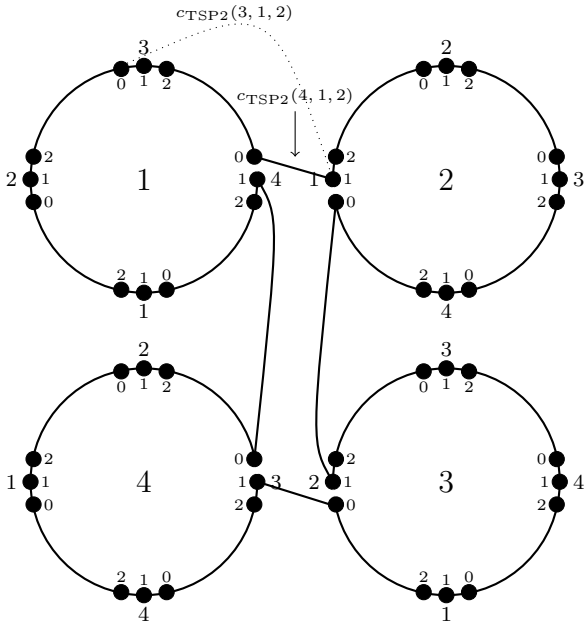


Figure 3: Example for $n = 4$ and tour $(1, 2, 3, 4)$

The set C_i is comprised of edges which form a cycle, and the vertex set of C_i is denoted by $V(C_i)$. With these notations, we are able to define the cost function c_{TSP}, where $i, i', j, j' \in N$ and $p, p' \in \{0, 1, 2\}$:

$$c_{\text{TSP}}(v_{i,j,p}, v_{i',j',p'}) = \begin{cases} 0, & \text{if } \{v_{i,j,p}, v_{i',j',p'}\} \in C_i \\ K + c_{\text{TSP2}}(j, i, i'), & \text{if } j' = i \neq i', p = 0, p' = 1 \\ K + c_{\text{TSP2}}(j', i', i), & \text{if } j = i' \neq i, p = 0, p' = 1 \\ 2K, & \text{otherwise.} \end{cases}$$

Idea: Each vertex $i \in V_{\text{TSP2}}$ corresponds to the vertex set $V(C_i)$ of the TSP instance and each $(i, j, k) \in (V_{\text{TSP2}})^3$ to the edge $\{v_{j,i,0}, v_{k,j,1}\}$. The cost function enforces that in an optimal solution each C_i is passed completely except for one edge after entering a vertex $v \in V(C_i)$ and that edges with costs $2K$ are not possible. Furthermore the vertices with $p = 2$ ensure that the tour enters C_i at vertex $v_{i,j,0}$ and leaves C_i at $v_{i,j,1}$ or vice versa (see Fig. 3 for illustration). We easily receive:

Remark 4.1. *For a TSP2 instance of size n the construction gives a TSP instance of size* $3n^2$.

W.l.o.g. we can consider only such cost functions with $c_{\text{TSP2}}(i, j, k) \geq 0 \, \forall i, j, k \in N$, as otherwise we could add an arbitrarily large constant to all costs and receive a TSP2 instance with equivalent optimal solutions.

Lemma 4.2. *Let π be a feasible solution of I_{TSP2}. Then a feasible solution π' of I_{TSP} can be constructed in polynomial time with*

$$c_{TSP}(\pi') = c_{TSP2}(\pi) + n \cdot K.$$

Proof. Let $\pi = (\pi(1), \ldots, \pi(n))$, where for $i \in N$ $\pi(i)$ denotes the vertex at position i in the tour. π' is created in the following way:

1 Start at vertex $v_{\pi(1),\pi(n),1}$ and set $i := 1, j := n$.
2 Add all edges $C_{\pi(1)} \setminus \{\{v_{\pi(1),\pi(n),0}, v_{\pi(1),\pi(n),1}\}\}$.
3 **WHILE** $i < n$
4 Set $j := i$.
5 Set $i := i + 1$.
6 Go to vertex $v_{\pi(i),\pi(j),1}$.
7 Add all edges $C_{\pi(i)} \setminus \{\{v_{\pi(i),\pi(j),0}, v_{\pi(i),\pi(j),1}\}\}$.
8 Go back to the starting point.

Then π' looks as follows:

$$\begin{aligned}\pi' = (&\underbrace{v_{\pi(1),\pi(n),1}, v_{\pi(1),\pi(n),2}, v_{\pi(1),(\pi(n) \bmod n)+1,0}, \ldots, v_{\pi(1),\pi(n),0}}_{C_{\pi(1)}},\\ &\underbrace{v_{\pi(2),\pi(1),1}, v_{\pi(2),\pi(1),2}, \ldots, v_{\pi(2),\pi(1),0}}_{C_{\pi(2)}}, v_{\pi(3),\pi(2),1}, \ldots,\\ &\underbrace{v_{\pi(n),\pi(n-1),1}, v_{\pi(n),\pi(n-1),2}, \ldots, v_{\pi(n),\pi(n-1),0}}_{C_{\pi(n)}}).\end{aligned}$$

It holds

$$\begin{aligned}c_{\mathrm{TSP}}(\pi') &= c_{\mathrm{TSP}}(v_{\pi(n),\pi(n-1),0}, v_{\pi(1),\pi(n),1}) + c_{\mathrm{TSP}}(v_{\pi(1),\pi(n),0}, v_{\pi(2),\pi(1),1})\\ &\quad + \sum_{i=1}^{n-2} c_{\mathrm{TSP}}(v_{\pi(i+1),\pi(i),0}, v_{\pi(i+2),\pi(i+1),1}) + 0\\ &= K + c_{\mathrm{TSP2}}(\pi(n-1), \pi(n), \pi(1)) + K + c_{\mathrm{TSP2}}(\pi(n), \pi(1), \pi(2))\\ &\quad + \sum_{i=1}^{n-2}(K + c_{\mathrm{TSP2}}(\pi(i), \pi(i+1), \pi(i+2)))\\ &= n \cdot K + c_{\mathrm{TSP2}}(\pi).\end{aligned}$$

□

Lemma 4.3. *Let π^* and π'^* be optimal solutions of I_{TSP2} and I_{TSP}, respectively. Then it holds:*

$$c_{TSP}(\pi'^*) \;\leq\; n \cdot K + c_{TSP2}(\pi^*) \;<\; (n+1) \cdot K.$$

Proof. This follows directly from Lemma 4.2 and $c_{\text{TSP2}}(\pi) < K$ for all tours π. □

Theorem 4.4. *TSP2 can be reduced in polynomial time to STSP, i.e., an optimal solution π^* of I_{TSP2} can be constructed by an optimal solution π'^* of I_{TSP}. More precisely, it holds*

$$c_{TSP2}(\pi^*) + n \cdot K = c_{TSP}(\pi'^*).$$

Proof. Let π'^* be an optimal solution of I_{TSP}. Since π'^* is feasible, at least n edges $\{v_{i,j,k}, v_{i',j',k'}\}$ exist with $i \neq i'$ (otherwise π'^* is not a tour over all vertices). As each of these edges has costs at least K, it follows with Lemma 4.3 that no edge with costs $2K$ can belong to π'^*. Next we show

$$|E(\pi'^*) \cap C_i| \;=\; 3n - 1 \,\forall i \in N.$$

On the one hand, if $|E(\pi'^*) \cap C_i| > 3n - 1$, π'^* would have a subtour in C_i. On the other hand, as π'^* is feasible, each set $V(C_i)$ is entered at least two times, which produces costs $c_{\text{TSP}}(\pi'^*) \geq \frac{2n \cdot K}{2} = n \cdot K$. Assume $|E(\pi'^*) \cap C_i| < 3n - 1$ for one $i \in N$. Then the set $V(C_i)$ is entered at least four times and it follows $c_{\text{TSP}}(\pi'^*) \geq \frac{2 \cdot (n-1) \cdot K + 4 \cdot K}{2} = (n+1) \cdot K$ which is a contradiction to Lemma 4.3.

Now we know that each set $V(C_i)$ is entered exactly two times and this does not happen at a vertex with $p = 2$, as no edge of the tour has costs $2K$, but at two vertices $v_{i,j,0}$ and $v_{i,j,1}$ for some $j \in N$. Let the tour π'^* start at $v_{1,j,1}$. Following the edges of C_1, the vertex $v_{1,j,0}$ is reached. Then the next vertex is $v_{k,1,1}$ etc. Thus we get a well-defined order π in which the C_i are visited, i.e., π'^* has the form of Lemma 4.2:

$$\begin{aligned}\pi'^* = (&\underbrace{v_{\pi(1),\pi(n),1}, v_{\pi(1),\pi(n),2}, v_{\pi(1),(\pi(n) \bmod n)+1,0}, \ldots, v_{\pi(1),\pi(n),0}}_{C_{\pi(1)}},\\ &\underbrace{v_{\pi(2),\pi(1),1}, v_{\pi(2),\pi(1),2}, \ldots, v_{\pi(2),\pi(1),0}}_{C_{\pi(2)}}, v_{\pi(3),\pi(2),1}, \ldots,\\ &\underbrace{v_{\pi(n),\pi(n-1),1}, v_{\pi(n),\pi(n-1),2}, \ldots, v_{\pi(n),\pi(n-1),0}}_{C_{\pi(n)}}).\end{aligned}$$

By Lemma 4.2, it follows

$$c_{\text{TSP}}(\pi'^*) \;=\; n \cdot K + c_{\text{TSP2}}(\pi). \tag{1}$$

Let π^* be an optimal solution of I_{TSP2}. Then we have

$$n \cdot K + c_{\text{TSP2}}(\pi^*) \leq n \cdot K + c_{\text{TSP2}}(\pi) \overset{(1)}{=} c_{\text{TSP}}(\pi'^*) \overset{\text{(Lemma 4.3)}}{\leq} n \cdot K + c_{\text{TSP2}}(\pi^*)$$

and it follows

$$c_{\text{TSP}}(\pi'^*) = n \cdot K + c_{\text{TSP2}}(\pi) = n \cdot K + c_{\text{TSP2}}(\pi^*)$$

as well as

$$c_{\mathrm{TSP2}}(\pi) = c_{\mathrm{TSP2}}(\pi^*)$$

which means π is an optimal solution of I_{TSP2}.

□

Clearly, all presented constructions are possible in polynomial time.

5 Heuristics for the TSP2

Let in the following for a given tour T and a given vertex $v \in T$, $p(v)$ be the predecessor of v and $s(v)$ be the successor of v.

5.1 Cheapest-Insert Algorithm

The *Cheapest-Insert Algorithm* (CI) is a generalization of an algorithm for the ATSP [29]. We start with an arc (v_1, v_2) considered as a subtour and choose this arc such that the term

$$\min_{x \in V} c(x, v_1, v_2) + \min_{x \in V} c(v_1, v_2, x)$$

is minimal. Then step by step, a new vertex is included in the subtour, so that the new subtour is cost minimal. If the tour is complete, we stop this procedure.

5.2 Nearest-Neighbor Algorithm

The *Nearest-Neighbor Algorithm* (NN) is also a generalization of an algorithm for the ATSP [29]. Again we start with one arc (v_1, v_2), now considered as a path. Then step by step, we compute neighbors in the direction $v_1 \to v_2$ in such a way that the new path becomes cost minimal. We stop, as soon as the path contains n vertices and we receive a tour. As we only walk in one direction, the predecessor of v_1 is chosen in the last step. As only one possibility for the predecessor of v_1 exists in this step, the costs $(p(p(p(v_1))), p(p(v_1)), p(v_1)))$, $(p(p(v_1)), p(v_1), v_1))$, and $(p(v_1), v_1, s(v_1))$ are irrelevant for this choice. Thus in the first step we choose the arc (v_1, v_2) in such a way that the term

$$\frac{1}{n-2} \cdot \left(\sum_{x \in V} c(x, v_1, v_2) \right) + \min_{x \in V} c(v_1, v_2, x)$$

is minimized.

5.3 Two-Directional-Nearest-Neighbor Algorithm

In this section we propose a variation of the Nearest-Neighbor Algorithm, which we call *Two Directional Nearest-Neighbor Algorithm* (2NN). For this algorithm, we contribute two important ideas. The first idea is to use both directions to find the next neighbor. Thus it is the question, which direction should be chosen in each step. One criterion for this choice is to use the minimal cost neighbor over all new vertices and over both directions. Our idea is based on the fact that the tour has to be closed anyway, so that *both* directions have to be used now or at a later step of the algorithm. Thus for a given path $(v_1, \ldots, v_i)$, the cost values $c(v_{i-1}, v_i, x)$ for a cost minimal neighbor vertex x and $c(y, v_1, v_2)$ for a cost minimal neighbor vertex y themselves are less important than the difference to the second smallest values in both directions. For both directions, this value can be viewed as an *upper tolerance* of the problem of finding a cost minimal neighbor vertex (for an overview over the theory of tolerances see [13, 14]). A similar idea was used for a tolerance based version [10] of the greedy heuristic [11] for the ATSP and a tolerance based version [15] of the Contract-or-Patch heuristic for the ATSP [11, 19]. Thus we choose the direction for which the upper tolerance value is larger, as not using the cost minimal neighbor vertex would cause a larger jump of the costs at a later step.

5.4 Assignment-Patching Algorithm

A well-known technique for the ATSP is the patching technique, which starts from a feasible AP solution. Then – step by step – it patches two cycles together, until there is only one cycle, which is the ATSP tour of this heuristic. As the optimal AP solution can be computed efficiently and the solution value is a good lower bound for an optimal ATSP solution value, it is a good starting point for patching. The corresponding AP instance to an ATSP instance uses the same cost function c with $c(v, v) = \infty$ for all $v \in V$. Karp and Steele suggested for each step to patch the two cycles containing the most number of vertices [25]. For each patching step for cycles C_1 and C_2, two arcs $e_1 \in C_1 = (v_1, w_1)$ and $e_2 = (v_2, w_2) \in C_2$ are replaced by arcs (v_1, w_2) and (v_2, w_1). These arcs are chosen in such a way from both cycles that we receive a minimal cost set of cycles in the next step. For the ATSP this means that the term

$$c(v_1, w_2) + c(v_2, w_1) - c(v_1, w_1) - c(v_2, w_2)$$

is minimized. We can transform this algorithm to a TSP2 algorithm, where for TSP2 the following term has to be minimized:

$$\begin{aligned} &c(p(v_1), v_1, w_2) + c(v_1, w_2, s(w_2)) + c(p(v_2), v_2, w_1) + c(v_2, w_1, s(w_1)) \\ &-c(p(v_1), v_1, w_1) - c(v_1, w_1, s(w_1)) - c(p(v_2), v_2, w_2) - c(v_2, w_2, s(w_2)). \end{aligned} \tag{2}$$

Analogously to the ATSP, the optimal AP2 solution would be a good starting point for patching, but by Theorem 3.2, AP2 is $\mathcal{NP}$-hard (in contrast to AP). One way to solve it, is by integer programming (see Section 6.2). However, this approach is not fast enough

for an efficient heuristic. Instead we propose to approximate an optimal AP2 solution by a polynomial time solvable heuristic solution. For this purpose, we define a two-dimensional cost function $c' : V \times V \to \mathbb{R} \cup \{\infty\}$, which depends on the three-dimensional cost function $c : V \times V \times V \to \mathbb{R}$ as follows: $c'(v, w) = \min_{u \in V} c(u, v, w)$. The AP solution of this cost function can be computed in $\mathcal{O}(n^3)$ and is a lower bound for the AP2 solution, which we call *approximated AP2 solution*. Then we patch the cycles of this AP solution, and we receive a feasible TSP2 solution. We call the approach *Assignment-Patching Algorithm* (AK), where "K" stands for "Karp Steele".

5.5 Nearest-Neighbor-Patching Algorithm

One property of the NN algorithm is that the number of remaining vertices becomes smaller (by 1) at each step. Thus on average the difference between the costs of the current path after adding one vertex and before should increase at each step. The idea of the following algorithm is to modify the NN algorithm in such a way that it outputs not a tour, but a set of cycles. Then these cycles are patched by the Patching Algorithm.

The main step of the *Nearest Neighbor Patching Algorithm* (NNK) is to stop the NN Algorithm, if closing the current cycle would lead to a "good" subtour. More precisely, we change the path $(v_1, \ldots, v_i)$ to a cycle, if the sum of the two costs $c(v_{i-1}, v_i, v_1)$ and $c(v_i, v_1, v_2)$, which are added by the closing, are smaller than a bound. Experiments have shown that $2 \cdot \sum_{j=1}^{i-2} c(v_j, v_{j+1}, v_{j+2})$ seems to be a good choice for this bound. As all cycles should contain at least 3 vertices and the rest of the graph has also to be divided into cycles, it requires that $3 \leq i \leq n-3$. We repeat these steps with the remaining vertices, until each vertex is contained in exactly one cycle.

5.6 Two Directional Nearest-Neighbor-Patching Algorithm

The *Two Directional Nearest Neighbor Patching Algorithm* (2NNK) is exactly the NNK Algorithm with the only difference that the 2NN Algorithm is used instead of the NN Algorithm for the computation of the cycles.

5.7 Greedy Algorithm

The *Greedy Algorithm* (G) is also a generalization of an ATSP algorithm [11] which is based on the contraction procedure. Let $G = (V, E)$ be a complete directed graph with $n \geq 3$ vertices and $c : E \to \mathbb{R}$ a cost function. Furthermore let an arbitrary arc e be given, w.l.o.g. $e = (v_{n-1}, v_n)$. The contraction of e means constructing a new complete graph $G' = (V', E')$ with $V' = \{v'_1, \ldots, v'_{n-1}\}$ and $v'_i = v_i$ for $i = 1, \ldots, n-2$, $v'_{n-1} = (v_{n-1}, v_n)$ and with cost function $c' : E' \to \mathbb{R}$ defined by

$$c'(v'_i, v'_j) = \begin{cases} c(v_i, v_j), & \text{if } 1 \leq i \neq j \leq n-2 \\ c(v_i, v_{n-1}), & \text{if } 1 \leq i \leq n-2, j = n-1 \\ c(v_n, v_j), & \text{if } i = n-1, 1 \leq j \leq n-2. \end{cases}$$

Analogously we define the contraction procedure for a three-dimensional cost function:

$$
\begin{aligned}
& c'(v'_i, v'_j, v'_k) \\
= & \begin{cases}
c(v_i, v_j, v_k), & \text{if } 1 \leq i, j, k \leq n-2, i \neq j, i \neq k, j \neq k \\
c(v_i, v_j, v_{n-1}), & \text{if } 1 \leq i \neq j \leq n-2, k = n-1 \\
c(v_i, v_{n-1}, v_n) + c(v_{n-1}, v_n, v_k), & \text{if } 1 \leq i \neq k \leq n-2, j = n-1 \\
c(v_n, v_j, v_k), & \text{if } 1 \leq j \neq k \leq n-2, i = n-1.
\end{cases}
\end{aligned}
$$

The greedy algorithm starts with contracting a "good" arc. We choose such an arc in the same way as in the CI Algorithm. Then we contract this arc, i.e., this arc appears in the final tour, and construct a graph with a vertex less. This step is repeated, until only three vertices remain. For this graph exactly two possible tours exist. We choose the smaller one of those, and finally we re-contract, i.e., all vertices are replaced by the paths which they consist of.

5.8 k-OPT Algorithm

The common characteristic of all previous algorithms is that in different ways they construct a tour. The first tour which is found is also the outputted tour. This is called a *construction heuristic*. In this section we present a so called *improvement heuristic*, i.e., it starts with a tour produced by a construction heuristic and improves it. For introducing the k-OPT algorithm [27] we need the following definition. Let a complete graph $G = (V, E)$, $|V| = n$ and a tour T be given, and let $k \leq n$. Furthermore let a (two-dimensional or three-dimensional) cost function be given. A k-OPT step changes T by omitting k arcs from the tour and adding k arcs not from the tour in such a way that the set of arcs is still a tour after the change. T is called *k-optimal*, if no r-OPT step with $r \leq k$ reduces the costs of the tour. Note that in general a k-optimal tour is not unique.

Each tour received by one of the previous construction heuristics can be transformed to a k-optimal tour by doing tour improving r-OPT steps with $r \leq k$, as long as they exist. With purpose to reduce the running time, k-OPT steps with smaller k are preferred. More precisely, the search starts with the smallest possible r which is $r = 3$ for the general case and is $r = 2$ for the special symmetric case (see Section 7.1). Then r increases by 1, until $r = k$. For fixed r, all sets of omitted arcs are traversed in a lexicographical order. For fixed r and for a fixed set of omitted arcs, *all* possible OPT steps for this choice are traversed (see again Section 7.1). If an improving OPT step has been found, the search continues again with the smallest possible r. As it is customary in literature [20], we consider only the case $k = 5$ in our experiments.

6 Exact Algorithms for the TSP2

6.1 Branch-and-Bound Algorithm

The following Branch-and-Bound Algorithm (BnB) visits in the worst case all possible tours in a lexicographic order and computes the tour with minimal costs. To avoid visiting all tours, it computes (local) lower bounds and upper bounds by visiting and analysing subpaths of all possible tours.

First we start with an arbitrary heuristic for the TSP2 to compute a good upper bound. Each time a new subpath is considered, a lower bound lb for a TSP2 solution containing this subpath is computed. As lower bound the approximated AP2 solution (see Section 5.4) is used. If lb is greater or equal than the current upper bound ub, we can prune this branch. The upper bound is updated, if a whole tour with smaller costs is visited. All tours are started with a fixed vertex v_1, which is chosen in such a way that the sum over all values $c(v_1, x, y)$ with $x \neq v_1, y \neq v_1, x \neq y$ is maximized. This choice is used, because we expect more prunes to appear, if the lower bounds in the first steps are rather large.

Pseudo-Code of the Branch-and-Bound Algorithm

1 Compute an upper bound ub for the TSP2 solution by a heuristic.
2 Choose a vertex $v_1 \in V$ in such a way that the following term is maximized:
$\sum_{x \in V \setminus \{v_1\}} \sum_{y \in V \setminus \{v_1, x\}} c(v_1, x, y)$.
3 Set solution value $p := ub$.
4 Let $v_2, \ldots, v_n$ be the remaining vertices.
5 Order all possible $(n-1)!$ tours $(v_1, v_{i_2}, v_{i_3}, \ldots, v_{i_n})$ lexicographically.
6 $i := 1$.
7 **IF** $i < n$
8 **THEN** Contract the current path from v_1 up to the i-th vertex to a new vertex and receive a new graph G'.
9 Compute the approximated AP2 solution of G' and receive a (local) lower bound lb.
10 **ELSE** $ub :=$ costs of the current tour.
11 **IF** $ub < p$
12 **THEN** $p := ub$.
13 **IF** $lb \geq p$ or $i = n$
14 **THEN** Search the largest j with $2 \leq j \leq n$ for which an unconsidered path $(v_1, w_2, \ldots, w_j)$ exists which is equal to the last path up to the $(j-1)$-th vertex.
15 **IF** No such j exists.
16 **THEN TERMINATE** with solution value p.
17 **ELSE** $i := j$.
18 **ELSE** Choose w_{i+1} in such a way that the path $(v_1, w_2, \ldots, w_i, w_{i+1})$ is the first one in the lexicographic order.

19 $i := i + 1$.
20 GOTO 7.

6.2 Integer-Programming Algorithm

The AP can be described by the following integer program IP_{AP}:

$$\min \sum_{i=1}^{n} \sum_{j=1, j\neq i}^{n} c_{ij} x_{ij} \tag{3}$$

$$\text{subject to} \sum_{j=1, j\neq i}^{n} x_{ij} = 1 \quad \forall 1 \leq i \leq n, \tag{4}$$

$$\sum_{i=1, i\neq j}^{n} x_{ij} = 1 \quad \forall 1 \leq j \leq n, \tag{5}$$

$$x_{ij} \in \{0, 1\} \quad \forall 1 \leq i \neq j \leq n, \tag{6}$$

where $C = (c_{ij})_{1 \leq i \neq j \leq n}$ is the cost matrix of the AP instance. Eq. (4) means that each vertex has exactly one out-arc and Eq. (5) that each vertex has exactly one in-arc. The AP solution consists of all arcs with $x_{ij} = 1$. The value c_{ij} of arc (i, j) is added to the objective value in Eq. (3), if and only if $x_{ij} = 1$.

As a natural generalization, the AP2 can be modelled by the following quadratic integer program QP_{AP2}:

$$\min \sum_{i=1}^{n} \sum_{j=1, j\neq i}^{n} \sum_{k=1, k\neq i, k\neq j}^{n} c_{ijk} x_{ij} x_{jk} \tag{7}$$

$$\text{subject to} \sum_{j=1, j\neq i}^{n} x_{ij} = 1 \quad \forall 1 \leq i \leq n, \tag{8}$$

$$\sum_{i=1, i\neq j}^{n} x_{ij} = 1 \quad \forall 1 \leq j \leq n, \tag{9}$$

$$x_{ij} \in \{0, 1\} \quad \forall 1 \leq i \neq j \leq n, \tag{10}$$

where $C = (c_{ijk})_{1 \leq i,j,k \leq n,\ i\neq j, i\neq k, j\neq k}$ is the cost matrix of the AP2 instance. Eq. (8) and (9) are the same as in the integer program for the AP. The value c_{ijk} of the sequence of vertices (i, j, k) is added to the objective value in Eq. (7), if and only if $x_{ij} = 1$ *and* $x_{jk} = 1$. As quadratic integer programs are harder to compute than integer programs [9], we need a corresponding integer program, so that each feasible solution of the integer program has a corresponding feasible solution of the quadratic integer program with the same objective value. This is called a linearization of the quadratic integer program. The

following linearization IP_{AP2} is similar to the linearization of the quadratic assignment problem (QAP) shown in [9]:

$$\min_{\text{subject to}} \sum_{i=1}^{n} \sum_{j=1,j\neq i}^{n} \sum_{k=1,k\neq i,k\neq j}^{n} c_{ijk} y_{ijk} \tag{11}$$

$$\sum_{j=1,j\neq i}^{n} x_{ij} = 1 \ \forall 1 \leq i \leq n, \tag{12}$$

$$\sum_{i=1,i\neq j}^{n} x_{ij} = 1 \ \forall 1 \leq j \leq n, \tag{13}$$

$$x_{ij} = \sum_{k=1,k\neq i,k\neq j}^{n} y_{ijk} \ \forall 1 \leq i \neq j \leq n, \tag{14}$$

$$x_{jk} = \sum_{i=1,i\neq j,i\neq k}^{n} y_{ijk} \ \forall 1 \leq j \neq k \leq n, \tag{15}$$

$$x_{ij} \in \{0,1\} \ \forall 1 \leq i \neq j \leq n, \tag{16}$$

$$y_{ijk} \in \{0,1\} \ \forall 1 \leq i,j,k \leq n, \ i \neq j, i \neq k, j \neq k. \tag{17}$$

In the objective function (11), the product $x_{ij} \cdot x_{jk}$ is replaced by the variable y_{ijk}, i.e., the value c_{ijk} of the sequence of vertices (i,j,k) is added to the objective value in (11), if and only if $y_{ijk} = 1$. Eq. (14) means that for each arc (i,j) in the AP2 solution there is exactly one sequence of vertices (i,j,k) starting with (i,j). Analogously, Eq. (15) means that for each arc (j,k) in the AP2 solution there is exactly one sequence of vertices (i,j,k) ending with (j,k).

To show that indeed IP_{AP2} is a linearization of QP_{AP2}, we have to ensure that

$$y_{ijk} = x_{ij} \cdot x_{jk} \quad \forall \ 1 \leq i,j,k \leq n, \ i \neq j, i \neq k, j \neq k. \tag{18}$$

First, let $x_{ij} = 0$. Then by Eq. (14) and (17) it follows $y_{ijk} = 0$, and thus Eq. (18) holds. The same holds for the case $x_{jk} = 0$ by using Eq. (15) and (17). It remains to consider the case $x_{ij} = x_{jk} = 1$. Assume $y_{ijk} = 0$. Then there exists $m \neq k$ with $y_{ijm} = 1$ by Eq. (14), and this implies $x_{j,m} = 1$ by Eq. (15). This is a contradiction to Eq. (12).

Remark 6.1. *The linearization IP_{AP2} keeps correct, if we replace the condition (17) by its linear relaxation, i.e., by*

$$0 \leq y_{ijk} \leq 1 \quad \forall 1 \leq i,j,k \leq n, \ i \neq j, i \neq k, j \neq k. \tag{19}$$

Proof. Assume i,j,k exist with $1 \leq i,j,k \leq n, \ i \neq j, i \neq k, j \neq k$ and $0 < y_{ijk} < 1$. Then it follows by Eq. (14) that there is a $k' \notin \{k,i,j\}$ with $0 < y_{ijk'} < 1$. From Eq. (15), we conclude $x_{jk} = 1, x_{jk'} = 1$. This contradicts Eq. (12), and we have $y_{ijk} \in \{0,1\}$. □

Note that after the replacement of Remark 6.1, the IP model IP_{AP2} should be easier to solve, as it contains less integer variables.

Now we come to the models for the TSP2. Like for the AP, an AP2 solution consists of $k \geq 1$ cycles. If $k = 1$, the AP2 solution is also a TSP2 solution. To avoid the possibility $k > 1$, it is sufficient that for each subset $S \subset V$ with $2 \leq |S| \leq n-2$ the solution contains at most $|S|-1$ arcs (i, j) with $i, j \in S$. Thus the following subtour elimination constraints (SEC) [6] can be added to the integer program IP_{AP2} leading to an integer program IP_{TSP2} for the TSP2.

$$\sum_{i,j \in S} x_{ij} \leq |S| - 1 \quad \forall S \subset V,\ 2 \leq |S| \leq n-2. \tag{20}$$

Unfortunately an exponential number of inequalities of that type exists. For this purpose, we solve the integer program IP_{AP2} and – step by step – add inequalities of that type which are violated (see the following pseudo-code).

Pseudo-Code of the IP Algorithm for the TSP2

1 Define IP_{AP2} by the conditions (11)-(16)+(19).
2 Solve IP_{AP2}.
3 Receive a solution with k cycles and a cycle C with the smallest number of vertices. Let $S = \{v_{s_1}, v_{s_2}, \ldots, v_{s_t}\}$ be the vertices of cycle C.
4 **IF** $k = 1$
5 **THEN** Output TSP2 solution C.
6 **ELSE** Add condition Eq. (20) for this S to IP_{AP2}.
7 **GOTO** 2.

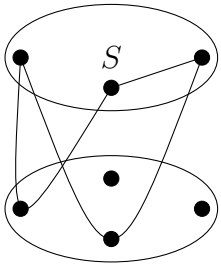
Furthermore we can use a good upper bound to speed-up the IP Algorithm.

6.3 Branch-and-Cut Algorithm

The Branch-and-Cut (BnC) technique is the most successful approach for exactly solving the STSP [6, 16, 28]. For instance, the BnC based TSP solver Concorde [3, 31] has recently solved a TSP instance of 85,900 cities [4], which up to date is the largest solved practical TSP instance [35].

In short words, a BnC algorithm is a BnB algorithm applied to an integer program, where linear relaxations of the integer program are used to receive lower bounds. To strengthen these lower bounds, inequalities are added, which are violated by the solution of the linear relaxation, but which must hold for the solution of the integer program. Such an inequality is called *cut* or *cutting plane*, and the non-trivial task of finding a cut is called *separation*. The relaxed program is repeatedly solved, until no cuts are found any more. Finally the branching part is applied, i.e., some variables are fixed and the procedure is applied to the sub-problems recursively, where again cuts can be used in the sub-problems.

Figure 4: Improved SEC cut



In this section we introduce a Branch-and-Cut Algorithm (BnC) for the TSP2 based on the ideas for the TSP. As a basis we use the integer program IP_{AP2}, where the condition (17) is replaced by its linear relaxation (see Remark 6.1). As already mentioned, an exponential number of SEC inequalities exists, and hence they cannot be added to the IP model at once. On the other hand, the SEC are needed to receive a TSP2 solution. Thus it is necessary to separate them as a cut. Fortunately, separating the SEC inequalities is possible in polynomial time (see e.g. [3, 21]). Using only SEC inequalities as cuts leads to the first version of our BnC Algorithm for the TSP2, which we call BnC-Pure or BnC-P for short. As the experiments in Section 8.2 will show, BnC-P is quite ineffective for the real instances. Therefore we present four further classes of cuts which essentially speed-up the BnC Algorithm in this case. We will denote the resulting algorithm by BnC-Extended or BnC-E for short.

1. The SEC inequalities (20) can be strengthened as follows:

$$\sum_{i,j\in S} x_{ij} + \sum_{i,j\in S, k\notin S} y_{ikj} \;\leq\; |S| - 1 \;\; \forall\, S \subset V,\; 2 \leq |S| < \frac{n}{2},$$

   where the subset S is required to have cardinality smaller than $n/2$. This strengthening is correct, as not only the direct connections between two vertices are counted, but also the connections with one vertex between them (see Fig. 4). As no easy polynomial-time separation algorithm exists for these cuts, we restrict to the case $|S| = 3$.

2. The following cuts hold for $|V| \geq 4$ and forbid all sub-cycles containing only 3 vertices:

$$y_{ijk} + y_{kij} \;\;\leq\;\; x_{ij} \;\; \forall\, 1 \leq i, j, k \leq n,\; i \neq j, i \neq k, j \neq k.$$

   As only $\mathcal{O}(n^3)$ such inequalities exist, they can be separated in polynomial time.

3. Let $i, j \in V$, $S \subset V \setminus \{i, j\}$, $T := V \setminus (S \cup \{i, j\})$ and $|V| \geq 5$. Then only one of the following variables can be set to 1, as otherwise we would have a cycle (see Fig. 5).

Figure 5: S-T-cut

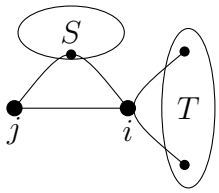
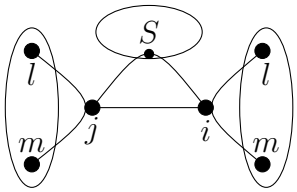


Figure 6: S-2-cut



- x_{ij} or x_{ji},
- y_{ikj} or $y_{jki}, k \in S$,
- $y_{lim}, l, m \in T$.

This leads to cuts of the form

$$x_{ij} + x_{ji} + \sum_{k \in S} (y_{ikj} + y_{jki}) + \sum_{l,m \in T} y_{lim} \leq 1. \tag{21}$$

Unfortunately, we have not found a polynomial-time separation algorithm for these cuts. Therefore we use the following heuristic approach to select an appropriate set S. For given $i, j \in V$ and a small constant $c > 0$ let $S := \{k \in V \setminus \{i, j\} \mid y_{ikj} + y_{jki} > c\}$. The idea is that for $k \in S$ the term $y_{i,k,j} + y_{jki}$ is large and for $k \notin S$ the term $\sum_{l \in V \setminus \{\{i,j,k\} \cup S\}} (y_{kil} + y_{lik})$ is greater than $y_{ikj} + y_{jki}$. By this choice of S, we hope that the left hand side of (21) is large.

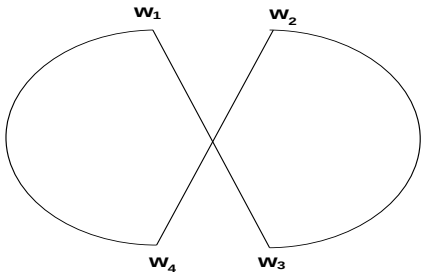
4. This cut is similar to the S-T-cut with $|T| = 2$ (see Fig. 6, where the set $T = \{l, m\}$ is drawn twice for better understanding). It is easy to see that Eq. (21) can be strengthened as follows:

$$x_{ij} + x_{ji} + \sum_{k \in S} (y_{ikj} + y_{jki}) + y_{lim} + y_{ljm} + y_{mil} + y_{mjl} \leq 1$$

since at most one of i and j can be in the middle of l and m. These cuts can also be separated in polynomial time.

Again we can use a good upper bound to speed-up the BnC Algorithm.

Figure 7: Only 2-OPT step for the STSP



6.4 TSP Reduction Algorithm

In Section 4 we have introduced a polynomial time reduction from a TSP2 instance to a TSP instance. Combining this reduction with an arbitrary TSP algorithm easily leads to a TSP2 algorithm. As this algorithm is based on the reduction to the TSP, we call it TSP-R.

7 Customization for Symmetric Instances

The main reason for the differentiation between the ATSP and the STSP is the fact that for the STSP specific STSP algorithms are used instead of general ATSP algorithms. This holds for both, heuristics (compare Helsgaun's LKH heuristic [20]) and exact algorithms (compare the solver Concorde [3, 31]). Note that both, LKH and Concorde, can also be applied to asymmetric instances by the 2-point reduction method, see [18, Chapter 2], [22].

For this reason we present two important specifications/optimizations of our proposed algorithms for symmetric instances, i.e., for instances of STSP2.

7.1 k-OPT Algorithm

For a given set of k arcs to be omitted from a tour, much more different sets of k arcs to be added to the tour exist for the STSP than for the ATSP. For example, consider Fig. 7. If the arcs (w_1, w_2) and (w_3, w_4) have been omitted from the tour, for the ATSP, there is no possibility to add 2 arcs not from the tour leading to a new tour, but for the STSP, adding (w_1, w_3) and (w_2, w_4) leads to a new tour. In Table 1, for $k = 2, 3, \ldots, 7$ the number of different k-OPT steps is listed, for both, ATSP and STSP.

Of course, this number does not change for the second order, as only the dimension of the cost function increases from 2 to 3, and the structure of the OPT steps keeps unchanged. Note that asymmetric k-OPT steps (k-A-OPT) can be applied to both, asymmetric instances and symmetric instances, but symmetric k-OPT steps (k-S-OPT) can be applied only to symmetric instances. Because of Table 1, we expect better upper bounds for symmetric instances by using S-OPT steps than by using A-OPT steps.

Table 1: Number of different k-OPT steps

k	ATSP	STSP
2	0	1
3	1	4
4	1	25
5	8	208
6	36	2121
7	229	25828

7.2 Integer-Programming Algorithm

In the symmetric case we can halve the number of variables and reduce the number of constraints by using only x_{ij} for $1 \leq i < j \leq n$ and y_{ijk} for $1 \leq i, j, k \leq n,\ i \neq j, j \neq k, i < k$. The adapted model for the IP_{AP2} reads

$$\min \sum_{i=1}^{n} \sum_{j=1, j\neq i}^{n} \sum_{k=1, k\neq j, k>i} c_{ijk} y_{ijk} \tag{22}$$

$$\text{subject to} \sum_{j=1, j>i}^{n} x_{ij} + \sum_{j=1, j<i}^{n} x_{ji} \ = \ 2 \ \ \forall\, 1 \leq i \leq n, \tag{23}$$

$$x_{ij} \ = \sum_{k=1, k\neq j, k>i}^{n} y_{ijk} + \sum_{k=1, k\neq j, k<i}^{n} y_{kji} \ \ \forall\, 1 \leq i < j \leq n, \tag{24}$$

$$x_{jk} \ = \sum_{i=1, i\neq j, i<k}^{n} y_{ijk} + \sum_{i=1, i\neq j, i>k}^{n} y_{kji} \ \ \forall\, 1 \leq j < k \leq n, \tag{25}$$

$$x_{ij} \in \{0, 1\} \ \ \forall\, 1 \leq i < j \leq n, \tag{26}$$

$$0 \leq y_{ijk} \leq 1 \ \ \forall\, 1 \leq i, j, k \leq n,\ i \neq j, j \neq k, i < k. \tag{27}$$

The cuts can be adapted in a similar way.

8 Experimental Study

We implemented all algorithms in C++, where we used the following subroutines for the TSP2 algorithms: for the BnB algorithm the AP solver implemented by Jonker and Volgenant [23, 34], which is based on the Hungarian method, for the IP algorithm the IP solver Cplex [32], and for the TSP-R algorithm the TSP solver Concorde [3, 31]. The implementation of the BnC algorithm is based on SCIP which is a C library providing the fundamental BnC framework and can be extended by problem specific plugins, e.g.,

the used cuts [1, 33]. All experiments were carried out on a PC with an Intel Xeon Dual Core CPU 3.0 GHz with 64 GB RAM. As test instances we chose four classes of random instances – two classes of asymmetric instances and two of symmetric instances – and three real (asymmetric) classes originating from the bioinformatics' application mentioned in the introduction. Let $C = (c_{ijk})_{1 \leq i,j,k \leq n}$ be the cost matrix of an instance, where $c_{ijk} = \infty$ for $i = j$ or $i = k$ or $j = k$. Then the four classes of random instances are defined as follows.

- Asymmetric Class 1:

  Each entry $c_{ijk} \neq \infty$ is independently chosen as an integer from $[0, \ldots, 10000]$.

- Symmetric Class 1:

  Each entry $c_{ijk} \neq \infty$ with $1 \leq i < k \leq n$ is independently chosen as an integer from $[0, \ldots, 10000]$, and c_{kji} is set to c_{ijk}.

- Asymmetric Class 2:

  Each entry $c_{ijk} \neq \infty$ is independently chosen as an integer from $[0, \ldots, i \cdot j \cdot k - 1]$.

- Symmetric Class 2:

  Each entry $c_{ijk} \neq \infty$ with $1 \leq i < k \leq n$ is independently chosen as an integer from $[0, \ldots, i \cdot j \cdot k - 1]$, and c_{kji} is set to c_{ijk}.

For the random instances we computed the average over 100 instances for the heuristics and 10 instances for the exact algorithms. All running times are given in seconds.

Additional information about the tested instances including the solution values received by the heuristics and the exact algorithms can be found at [36].

8.1 Comparison of Heuristics

An experimental study of heuristics for the ATSP is given in [18, Chapter 10]. In this section we make a similar study for the TSP2. In detail, we compare all considered heuristics, which are Cheapest-Insert Algorithm (CI), Nearest-Neighbor Algorithm (NN), Two-Directional Nearest-Neighbor Algorithm (2NN), Assignment-Patching Algorithm (AK), Nearest-Neighbor-Patching Algorithm (NNK), Two-Directional Nearest-Neighbor-Patching Algorithm (2NNK) and Greedy Algorithm (G). Furthermore we consider for each algorithm a version, where the algorithm is followed by the 5-A-OPT Algorithm (abbreviated by "+A"), and for the symmetric classes 1 and 2 we consider also a version, where the algorithm is followed by the 5-S-OPT Algorithm (abbreviated by "+S").

The most important criterion for the quality of a heuristic for a given instance is the excess of its upper bound over its optimal value. From this reason, we used in our experiments only instances with smaller size so that in most cases one of the exact algorithms is able to compute the optimal value. In particular we chose sizes 10, 20, 30, 40, 50 for the random classes and sizes 20, 40, 60, 80 for the real classes. The results for the asymmetric classes 1 and 2, for the symmetric classes 1 and 2, and the real classes 1, 2 and 3 can be found in Tables 2-7.

Running Times of the Heuristics

All basic algorithms are rather fast for both, the random and the real instances, which is reasonable, as they have complexity not worse than $\mathcal{O}(n^3)$. Only the times for G are slightly larger than those for the remaining versions. Comparing the algorithms with and without A-OPT steps, the algorithms with A-OPT steps are considerably slower. For the two symmetric classes, the S-OPT steps are considerably slower than the A-OPT steps.

Quality of the Tours Generated by the Heuristics

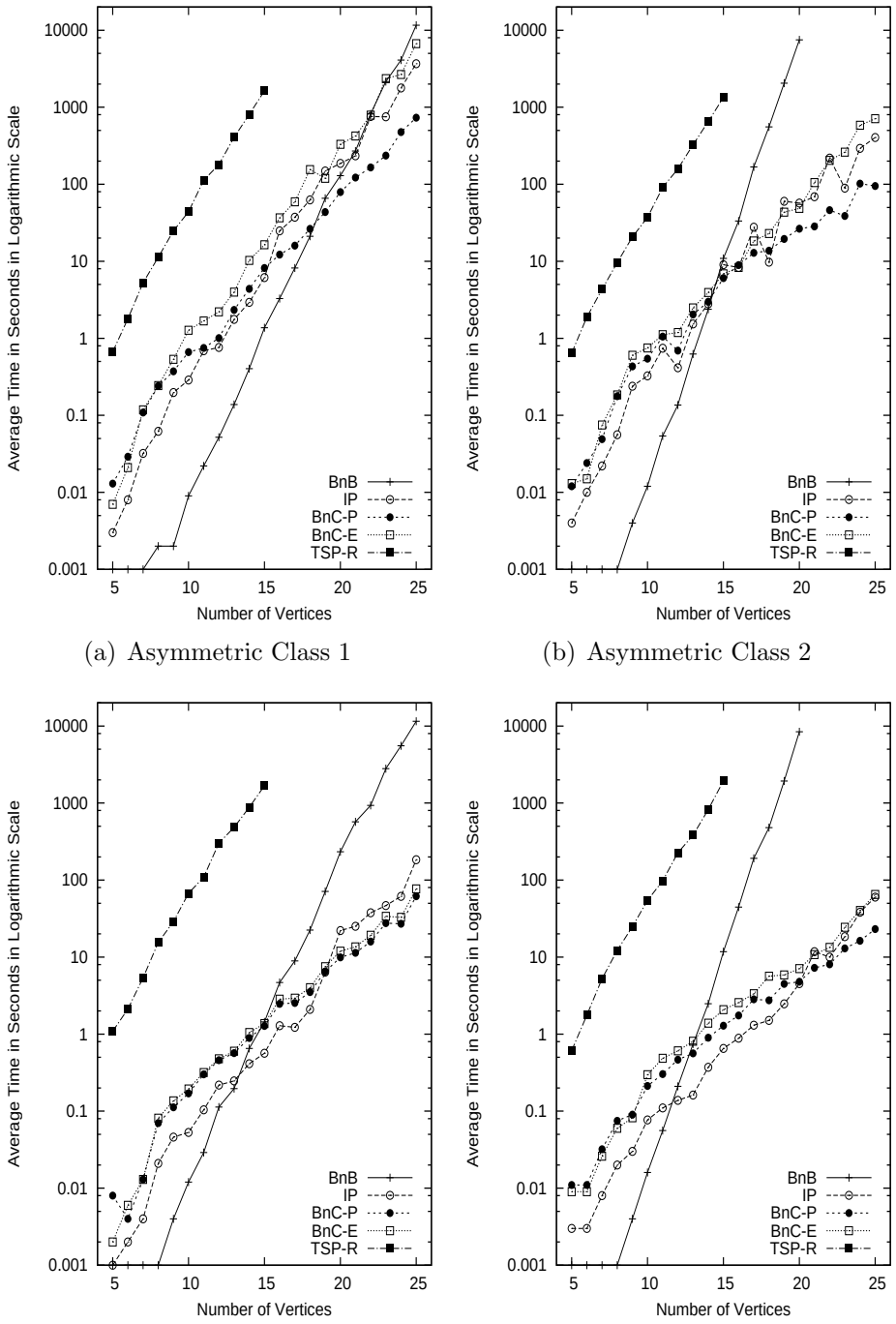
The experiments show that – as expected – the OPT versions clearly beat the basic versions and (for the symmetric classes 1 and 2) the S-OPT versions in average beat the A-OPT versions. Considering the basic versions, the results for the (asymmetric and symmetric) random classes 1, the (asymmetric and symmetric) random classes 2 and the real classes are completely different. For the random classes 1, 2NNK is the best algorithm, for the random classes 2, CI is the best algorithm, whereas for the real classes in average AK is the best algorithm. AK is the worst algorithm for random instances of class 1, and G is the worst algorithm for the remaining classes. Considering the OPT versions we observe the following. For the random instances 1, 2NNK is the best algorithms, and AK is the worst algorithm, whereas for the real classes, NN, NNK, 2NN and 2NNK are the best algorithms, and G is the worst algorithm. For the random instances 2, the results are mixed. Surprisingly, for these instances, G (which is the worst algorithm for the basic versions) is the best one in average. This could mean that sometimes the OPT versions benefit from worse starting tours.

Regarding the comparison with the exact values, we have to realize that the heuristics presented in this paper do not behave very well applied to random instances. They generate tours whose costs are considerably larger than the costs of optimal tours. Considering real instances, things look nicer. For 11 of the 12 instances at least one of the seven A-OPT versions finds the optimum. For the single remaining instance as well as for the unsuccessful A-OPT versions we receive upper bounds very close to the optimum.

8.2 Comparison of Exact Algorithms

In this section we compare the running times of all introduced five exact algorithms, namely the BnB Algorithm, the IP Algorithm, the BnC Algorithms BnC-P and BnC-E, and the TSP Algorithm TSP-R. The results can be found in Fig. 8 and Fig. 9. In the following we describe for each algorithm the main observations and give a short analysis.

- **BnB:** For small sizes, this algorithm is leading, but for large sizes it becomes the second-worst algorithm.
- **IP:** For the random instances, IP is competitive even with the best algorithm, namely BnC-P, whereas for the real instances it is considerably worse. To find an explanation for the behavior that the running times are better for random instances than for real



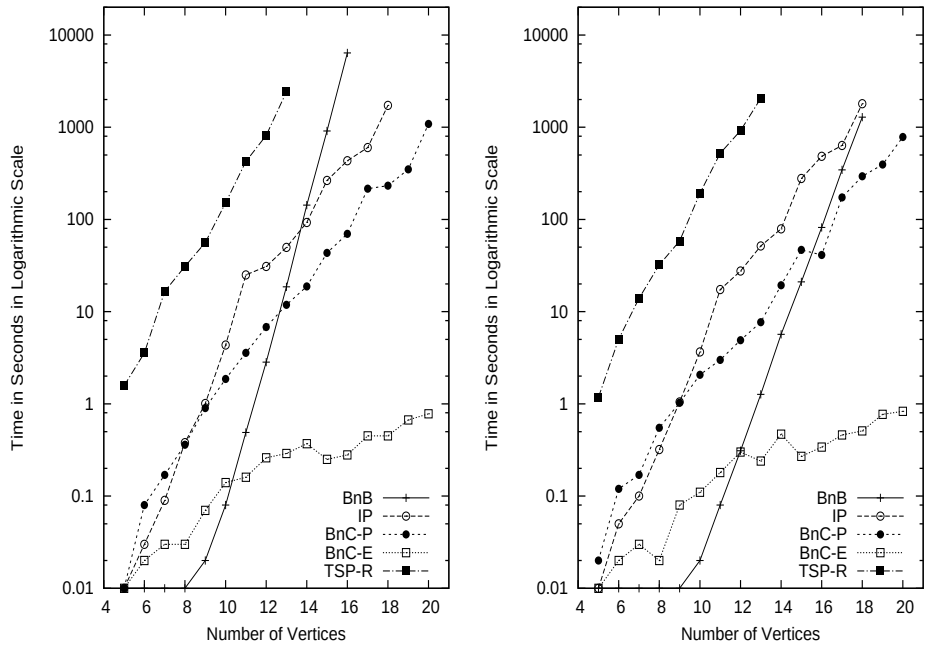
(a) Asymmetric Class 1

(b) Asymmetric Class 2

(c) Symmetric Class 1

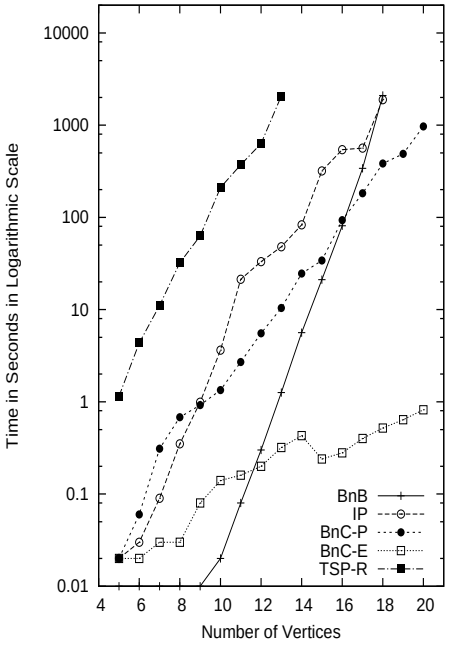
(d) Symmetric Class 2

Figure 8: Time comparison for exact algorithms applied to the random classes

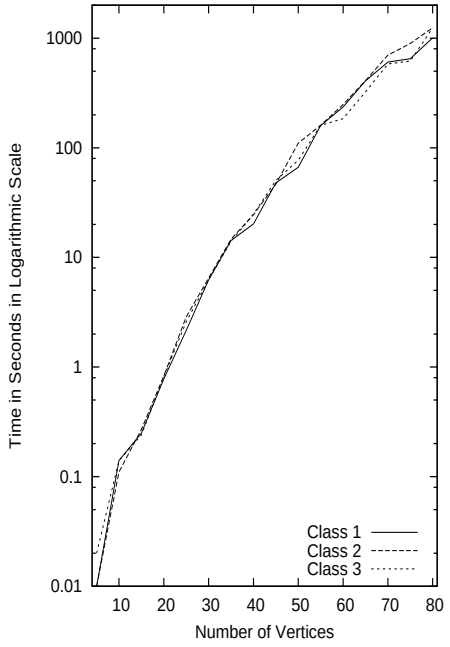


(a) Real Class 1

(b) Real Class 2



(c) Real Class 3



(d) Real Classes with BnC-E

Figure 9: Time comparison for exact algorithms applied to the real classes

instances, we consider the number of iterations of adding equalities of type (20) as a possible criterion. Indeed, the number of iterations is rather small for random instances: at most 9 iterations for 829 of the 840 instances of sizes $5, 6, \ldots, 25$. In contrast, for real instances this number is very large: at least 10 iterations for all 48 instances of sizes $10, 11, \ldots, 25$. Thus this number seems to be an important criterion for the running time.

- **BnC:** Comparing both BnC versions, namely the pure version BnC-P and the extended version BnC-E, we observe a completely different behavior for random and real instances. For random instances, BnC-P is the best algorithm and much faster than BnC-E. On the other hand, for real instances BnC-E runs with orders of magnitude faster. In particular, BnC-E is able to solve all real instances up to dimension 80 (see Fig. 9(d)). This time difference can be explained as follows. As it becomes clear from the results of the heuristics (see Section 8.1), a good (and often optimal) solution can be found very soon for the real instances. Thus the branching part in the BnC algorithm is not called very often. Nevertheless the (rarely called) computation of the cutting planes of the BnC-E algorithm are very helpful to prove the optimality. On the other hand, for random instances the value of the first linear relaxation is mostly far away from the optimal solution value leading to many branching steps. In these branching steps, the computation of the cutting planes is not able to do an essential reduction of the branching tree, but it costs much time.

- **TSP-R:** For all classes, the TSP-R algorithm is absolutely not competitive and is only able to solve small instances up to dimension 15. Note that the quality of TSP-R corresponds to the performance of Concorde applied to the TSP instance, which was constructed by the given TSP2 instance. For a TSP2 instance of size 15 this leads to a TSP instance of size $3 \cdot 15^2 = 675$ (see Remark 4.1). At first sight, this bad performance of Concorde is surprising, as Concorde has recently solved even a TSP instance of size $85,900$ [4]. We suppose that the structure of the considered TSP instances, which are neither Euclidean instances nor random instances, is rather bad for the application of Concorde, as it contains a large number of solutions with value close to the solution value. Furthermore because of the large constant K, many entries of the TSP instances are rather large, which could also be a problem for Concorde.

9 Summary and Future Research

The purpose of this paper is to introduce two new combinatorial optimization problems, namely TSP2 and AP2, where the TSP2 has important applications in bioinformatics. We show the $\mathcal{NP}$-hardness of both problems, propose seven heuristics and four exact algorithms for the TSP2 and compare them in an experimental study. For the real instances, the best of our heuristics finds the optima in almost all cases. Our main result is that

one of the exact algorithms, namely BnC-E, solves all real instances of our bioinformatics' application up to dimension 80 in reasonable time.

Both, from the theoretical point of view and for applications in bioinformatics, also the natural generalization of the TSP2 to larger orders, i.e., to TSPk for $k \geq 3$, seems quite interesting.

Acknowledgment

The second and fourth authors' work was supported by German Research Foundation (DFG) under grant number MO 645/7-3.

References

[1] T. Achterberg: Constraint Integer Programming. PhD Thesis, Technical University Berlin, Germany, 2007.

[2] A. Aggarwal, D. Coppersmith, S. Khanna, R. Motwani, B. Schieber: The Angular-Metric Traveling Salesman Problem. *Eighth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 221-229, 1997.

[3] D.L Applegate, R.E. Bixby, V. Chvátal, W.J. Cook: The Traveling Salesman Problem. A Computational Study. *Princeton University Press*, 2006.

[4] D.L Applegate, R.E. Bixby, V. Chvátal, W.J. Cook, D. Espinoza, M. Goycoolea, K. Helsgaun: Certification of an Optimal Tour through 85,900 Cities. *Oper. Res. Lett.* **37**(1), 11-15, 2009.

[5] D.P. Bertsekas: A New Algorithm for the Assignment Problem. *Math. Program.* **21**, 152-171, 1981.

[6] G. Dantzig, R. Fulkerson, S. Johnson: Solution of a Large-Scale Traveling-Salesman Problem. *Oper. Res.* **2**(4), 393-410, 1954.

[7] M. Dell'Amico, P. Toth: Algorithms and Codes for Dense Assignment Problems: the State of the Art. *Discrete Appl. Math.* **100**(1-2), 17-48, 2000.

[8] K. Ellrott, C. Yang, F.M. Sladek, T. Jiang: Identifying Transcription Factor Binding Sites Through Markov Chain Optimization. *Bioinformatics* **18**, 100-109, 2002.

[9] A.M. Frieze, J. Yadegar: On the Quadratic Assignment Problem. *Discrete Appl. Math.* **5**, 89-98, 1983.

[10] D. Ghosh, B. Goldengorin, G. Gutin, G. Jäger: Tolerance-Based Greedy Algorithms for the Traveling Salesman Problem. Chapter 5 in: Mathematical Programming and Game Theory for Decision Making. S.K. Neogy, R.B. Bapat, A.K. Das, T. Parthasarathy (Eds.). *World Scientific*, New Jersey, 47-59, 2008.

[11] F. Glover, G. Gutin, A. Yeo, A. Zverovich: Construction Heuristics for the Asymmetric TSP. *European J. Oper. Res.* **129**, 555-568, 2001.

[12] A.V. Goldberg, R. Kennedy: An Efficient Cost Scaling Algorithm for the Assignment Problem. *Math. Program.* **71**, 153-177, 1995.

[13] B. Goldengorin, G. Jäger, P. Molitor: Some Basics on Tolerances. In S.-W. Cheng, C.K. Poon (Eds.), *The Second International Conference on Algorithmic Aspects in Information and Management (AAIM)*. Lecture Notes in Comput. Sci. **4041**, 194-206, 2006.

[14] B. Goldengorin, G. Jäger, P. Molitor: Tolerances Applied in Combinatorial Optimization. *J. Comput. Sci.* **2**(9), 716-734, 2006.

[15] B. Goldengorin, G. Jäger, P. Molitor: Tolerance Based Contract-or-Patch Heuristic for the Asymmetric TSP. In T. Erlebach (Ed.), *The Third Workshop on Combinatorial and Algorithmic Aspects of Networking (CAAN)*. Lecture Notes in Comput. Sci. **4235**, 86-97, 2006.

[16] M. Grötschel, O. Holland: Solution of Large-Scale Symmetric Travelling Salesman Problems. *Math. Program.* **51**(2), 141-202, 1991.

[17] I. Grosse, J. Keilwagen, University of Halle-Wittenberg, Chair for Bioinformatics, and Leibniz Institute of Plant Genetics and Crop Plant Research in Gatersleben. Private communication.

[18] G. Gutin, A.P. Punnen (Eds.): The Traveling Salesman Problem and Its Variations. *Kluwer*, Dordrecht, 2002.

[19] G. Gutin, A. Zverovich: Evaluation of the Contract-or-Patch Heuristic for the Asymmetric TSP. *INFOR* **43**(1), 23-31, 2005.

[20] K. Helsgaun: An Effective Implementation of the Lin-Kernighan Traveling Salesman Heuristic. *European J. Oper. Res.* **126**(1), 106-130, 2000.

[21] S. Hong: A Linear Programming Approach for the Traveling Salesman Problem. PhD Thesis, Technical University Berlin, Germany, 1972.

[22] R. Jonker, A. Volgenant: Transforming Asymmetric into Symmetric Traveling Salesman Problems. *Oper. Res. Lett.* **2**(4), 161-163, 1983

[23] R. Jonker, A. Volgenant: A Shortest Augmenting Path Algorithm for Dense and Sparse Linear Assignment Problems. *Computing* **38**, 325-340, 1987.

[24] R.M. Karp: Reducibility Among Combinatorial Problems. In: *Complexity of Computer Computations*. R.E. Miller, J.W. Thatcher (Eds.). *New York: Plenum*, 85-103, 1972.

[25] R.M. Karp, J.M. Steele: Probabilistic Analysis of Heuristics. Chapter 6 in: The Traveling Salesman Problem. E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, D.B. Shmoys (Eds.). *John Wiley & Sons*, 181-205, 1985.

[26] H.W. Kuhn: The Hungarian Method for the Assignment Problem. Naval Res. Logist. Quarterly **2**, 83-97, 1955.

[27] S. Lin, B.W. Kernighan: An Effective Heuristic Algorithm for the Traveling-Salesman Problem. *Oper. Res.* **21**, 498-516, 1973.

[28] M. Padberg, G. Rinaldi: A Branch-and-Cut Algorithm for the Resolution of Large-Scale Symmetric Traveling Salesman Problems. *SIAM Rev.* **33**(1), 60-100, 1991.

[29] D.J. Rosenkrantz, R.E. Stearns, P.M. Lewis: An Analysis of Several Heuristics for the Traveling Salesman Problem. *SIAM J. Comput.* **6**, 563-581, 1977.

[30] X. Zhao, H. Huang, T.P. Speed: Finding Short DNA Motifs Using Permuted Markov Models. *Journal of Computational Biology* **12**, 894-906, 2005.

[31] Source code of [3] (Concorde). Available:
"http://www.tsp.gatech.edu/concorde.html".

[32] Homepage of CPLEX:
"http://www.ilog.com/products/optimization/archive.cfm".

[33] Source code of SCIP. Available:
"http://scip.zib.de/download.shtml".

[34] Source code of [23]. Available:
"http://www.magiclogic.com/assignment.html".

[35] TSP Homepage: "http://www.tsp.gatech.edu/".

[36] Additional information about the tested instances of this paper:
"http://www.informatik.uni-halle.de/ti/forschung/toleranzen/tsp2/".

	Asymmetric Class 1					Asymmetric Class 2				
Size	10	20	30	40	50	10	20	30	40	50
CI	21496	32371	41397	48974	56148	263	2471	9354	24988	53265
NN	26340	32615	37712	40470	43648	544	5779	23002	62157	125408
2NN	24422	31349	36030	39361	41502	518	6044	23429	65976	139659
AK	34208	71121	112836	151693	196960	453	6045	28383	93502	236753
NNK	22452	29211	35474	41509	45511	348	3555	14362	38676	79407
2NNK	23383	29167	32905	37395	40125	423	3908	14675	41325	80450
G	26695	44379	60912	73472	84602	561	7273	34391	104371	231228
CI+A	14802	19981	24327	28378	32150	142	1159	4261	10630	22038
NN+A	14945	19842	23012	26350	28486	143	1154	4041	10665	21793
2NN+A	14841	19582	22934	25801	27831	144	1170	4224	10491	21822
AK+A	14830	19995	24877	28882	32180	141	1150	4288	10496	21821
NNK+A	14836	19394	23057	25836	29588	146	1154	4305	10537	22578
2NNK+A	14667	18852	22693	26106	28365	141	1155	4103	10295	21233
G+A	14991	19622	23601	28234	31213	147	1153	4133	10183	21066
Exact	12403	11606	10998			120	705	2144		

Table 2: Quality comparison for heuristics applied to the asymmetric classes

	Asymmetric Class 1					Asymmetric Class 2				
Size	10	20	30	40	50	10	20	30	40	50
CI	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
NN	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
2NN	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
AK	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
NNK	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
2NNK	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
G	0.00	0.00	0.00	0.01	0.03	0.00	0.00	0.00	0.01	0.03
CI+A	0.00	0.02	0.24	1.24	4.74	0.00	0.02	0.25	1.33	5.11
NN+A	0.00	0.02	0.21	1.05	3.19	0.00	0.02	0.27	1.28	4.24
2NN+A	0.00	0.02	0.19	0.97	3.59	0.00	0.02	0.26	1.35	4.47
AK+A	0.00	0.02	0.25	1.30	4.80	0.00	0.02	0.25	1.48	5.19
NNK+A	0.00	0.02	0.22	1.24	3.62	0.00	0.02	0.22	1.20	4.07
2NNK+A	0.00	0.02	0.22	0.99	3.53	0.00	0.02	0.24	1.29	4.60
G+A	0.00	0.03	0.22	1.37	4.52	0.00	0.02	0.23	1.52	4.78

Table 3: Time comparison for heuristics applied to the asymmetric classes

	Symmetric Class 1					Symmetric Class 2				
Size	10	20	30	40	50	10	20	30	40	50
CI	22231	32730	41990	48407	55956	264	2453	9526	24775	53023
NN	25370	33246	37677	41775	42630	551	5646	22316	60593	125539
2NN	23698	30793	35535	39328	40694	542	6400	24035	62148	134717
AK	34247	73756	112530	154579	196941	490	5632	30877	97347	239418
NNK	22829	30195	37487	41579	45831	390	3651	14852	39812	83239
2NNK	23174	28583	33744	37890	41229	460	4015	14649	36839	77477
G	27706	45251	60205	73056	85281	542	7345	35107	103627	234193
CI+A	15546	19926	24378	28551	32179	152	1188	4273	10755	21811
NN+A	15591	19740	23591	26113	28655	154	1158	4244	10785	21949
2NN+A	15723	19663	22587	25583	27561	155	1172	4127	10663	20671
AK+A	15607	20654	24744	28526	32100	152	1229	4316	10510	22416
NNK+A	15689	19821	23566	26850	29502	155	1217	4205	10555	21494
2NNK+A	15544	19687	22908	25645	28862	150	1203	4147	10201	21557
G+A	15536	19489	23742	27672	31675	155	1187	4033	10487	21715
CI+S	16183	19798	23617	27599	30906	158	1201	4280	10483	21995
NN+S	15687	19849	22497	25063	27084	155	1189	4214	10182	21084
2NN+S	15835	19453	22065	24615	26231	159	1203	4180	10336	20912
AK+S	15763	20130	23711	27205	31098	157	1194	4152	10559	21879
NNK+S	16040	19472	22434	25577	28103	151	1203	4096	10305	21626
2NNK+S	16102	19120	21948	24466	27322	158	1217	4142	10078	20915
G+S	15775	19108	22196	26577	28998	158	1156	3989	10151	20599
Exact	13840	12380	11578			137	784	2249		

Table 4: Quality comparison for heuristics applied to the symmetric classes

	Symmetric Class 1					Symmetric Class 2				
Size	10	20	30	40	50	10	20	30	40	50
CI	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
NN	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
2NN	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
AK	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
NNK	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
2NNK	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
G	0.00	0.00	0.00	0.01	0.03	0.00	0.00	0.00	0.01	0.03
CI+A	0.00	0.02	0.24	1.30	4.68	0.00	0.02	0.25	1.35	4.42
NN+A	0.00	0.02	0.20	1.00	3.48	0.00	0.02	0.23	1.17	4.24
2NN+A	0.00	0.02	0.18	1.05	3.09	0.00	0.02	0.25	1.24	4.47
AK+A	0.00	0.02	0.23	1.33	4.66	0.00	0.02	0.23	1.37	4.95
NNK+A	0.00	0.02	0.20	1.02	3.95	0.00	0.02	0.23	1.33	4.83
2NNK+A	0.00	0.02	0.19	1.09	3.54	0.00	0.02	0.23	1.32	4.83
G+A	0.00	0.02	0.25	1.29	4.13	0.00	0.02	0.25	1.26	4.65
CI+S	0.00	0.08	1.19	7.23	26.61	0.00	0.08	0.96	6.41	24.84
NN+S	0.00	0.07	0.95	5.49	22.39	0.00	0.07	0.84	6.53	23.96
2NN+S	0.00	0.07	0.95	5.64	21.63	0.00	0.08	0.95	7.00	25.72
AK+S	0.00	0.06	1.03	6.67	26.00	0.00	0.08	1.03	6.46	24.57
NNK+S	0.00	0.07	0.87	6.13	21.46	0.00	0.07	0.95	5.88	29.03
2NNK+S	0.00	0.07	0.92	5.74	24.36	0.00	0.08	0.80	5.88	25.52
G+S	0.00	0.08	1.11	6.35	28.41	0.00	0.07	0.91	6.04	25.74

Table 5: Time comparison for heuristics applied to the symmetric classes

	Real Class 1				Real Class 2				Real Class 3			
Size	20	40	60	80	20	40	60	80	20	40	60	80
CI	2454	5315	8294	10981	2199	4483	6808	8877	2064	4344	6662	8727
NN	2490	5310	8276	10924	2243	4470	6786	8819	2101	4326	6640	8667
2NN	2490	5310	8276	10924	2243	4470	6786	8819	2101	4326	6640	8667
AK	2601	5288	8433	11131	2188	4432	6860	8785	2046	4292	6716	8633
NNK	2607	5775	9121	12425	2351	4873	7449	10201	2212	4732	7304	10053
2NNK	2652	5793	9156	12515	2367	4817	7951	10372	2231	4677	7699	10230
G	3026	6172	10352	13824	2737	5231	8224	11285	2629	5092	8182	10984
CI+A	2407	5175	8138	10793	2155	4330	6642	8682	2016	4189	6499	8533
NN+A	2407	5175	8138	10793	2155	4330	6642	8682	2016	4188	6498	8532
2NN+A	2407	5175	8138	10793	2155	4330	6642	8682	2016	4188	6498	8532
AK+A	2407	5243	8301	10797	2155	4408	6709	8679	2016	4266	6565	8529
NNK+A	2407	5175	8210	10998	2154	4329	6731	8910	2015	4188	6583	8859
2NNK+A	2491	5243	8246	10841	2154	4329	6641	8679	2015	4188	6653	8528
G+A	2439	5244	8354	10957	2185	4401	6713	8888	2047	4255	6651	8656
Exact	2407	5175	8138	10793	2154	4329	6641	8679	2015	4188	6494	8528

Table 6: Quality comparison for heuristics applied to the real classes

	Real Class 1				Real Class 2				Real Class 3			
Size	20	40	60	80	20	40	60	80	20	40	60	80
CI	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.01	0.00	0.00	0.00	0.01
NN	0.00	0.00	0.00	0.01	0.00	0.00	0.00	0.00	0.00	0.01	0.00	0.01
2NN	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.00	0.00	0.00	0.01
AK	0.00	0.00	0.00	0.01	0.00	0.00	0.01	0.00	0.00	0.00	0.00	0.01
NNK	0.00	0.00	0.02	0.03	0.00	0.00	0.01	0.03	0.00	0.00	0.00	0.02
2NNK	0.00	0.00	0.01	0.02	0.00	0.00	0.01	0.01	0.00	0.00	0.00	0.02
G	0.00	0.02	0.04	0.19	0.00	0.01	0.05	0.18	0.00	0.01	0.06	0.15
CI+A	0.01	0.33	2.53	11.18	0.01	0.30	2.53	11.07	0.00	0.32	2.54	11.12
NN+A	0.01	0.30	2.55	11.12	0.01	0.31	2.54	11.08	0.00	0.30	2.52	11.12
2NN+A	0.01	0.31	2.55	11.13	0.00	0.30	2.52	11.08	0.01	0.31	2.53	11.20
AK+A	0.01	0.30	2.59	11.28	0.01	0.91	2.53	11.07	0.01	0.91	2.57	11.12
NNK+A	0.01	0.31	7.67	44.37	0.01	1.20	7.64	33.42	0.01	1.20	7.80	33.10
2NNK+A	0.01	0.31	2.65	11.70	0.03	0.30	12.66	11.15	0.03	0.94	7.70	11.12
G+A	0.02	0.31	2.65	11.62	0.00	1.21	10.09	33.65	0.01	0.31	2.67	55.31

Table 7: Time comparison for heuristics applied to the real classes