Algorithms and Experimental Study for the Traveling Salesman Problem of Second Order

Gerold Jäger and Paul Molitor

Computer Science Institute,
University of Halle-Wittenberg,
D-06099 Halle (Saale), Germany
E-mail: jaegerg@informatik.uni-halle.de,
paul.molitor@informatik.uni-halle.de

Abstract. We introduce a new combinatorial optimization problem, which is a generalization of the Traveling Salesman Problem (TSP) and which we call Traveling Salesman Problem of Second Order (2-TSP). It is motivated by an application in bioinformatics, especially the Permuted Variable Length Markov model. We propose seven elementary heuristics and two exact algorithms for the 2-TSP, some of which are generalizations of similar algorithms for the Asymmetric Traveling Salesman Problem (ATSP), some of which are new ideas. Finally we experimentally compare the algorithms for random instances and real instances from bioinformatics. Our experiments show that for the real instances most heuristics lead to optimum or almost-optimum solutions, and for the random instances the exact algorithms need less time than for the real instances.

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

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 [5] for the recognition of transcription factor binding sites. The class of PM models was further extended by [24] 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 [5] and [24], respectively. Experimental evidence has been accumulated that suggests that the binding sites of many transcription factors fall into distinct classes. Grosse 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 [15]. 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 consist 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 an ATSP 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 ATSP, which we call 2-TSP and which is introduced in the following.

For a directed graph $G = (V, E)$ with $n \geq 2$ vertices and a weight function $c : V \times V \to \mathbb{R} \cup \{\infty\}$ with $c(u, u) = \infty$ for all $u \in V$ the *Asymmetric Traveling Salesman Problem* is the problem of finding a complete tour $(v_1, v_2, \ldots, v_n)$ with minimum costs $c(v_n, v_1) + \sum_{j=1}^{n-1} c(v_j, v_{j+1})$. ATSP 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 [20]. The special case that the weight of each arc equals the weight of the corresponding reverse arc is called *Symmetric Traveling Salesman Problem* (STSP).

In this paper we introduce the following problem, which – to the best of our knowledge – has not been considered in literature before. For a directed graph $G = (V, E)$ with $n \geq 3$ vertices and a weight 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 $v = w$ consider the problem of finding a complete tour $(v_1, v_2, \ldots, v_n)$ with minimum costs $c(v_{n-1}, v_n, v_1) + c(v_n, v_1, v_2) + \sum_{j=1}^{n-2} c(v_j, v_{j+1}, v_{j+2})$. As this problem is a natural generalization of the ATSP, where the costs do not depend on arcs, but on each sequence of three vertices in the tour, we call it *Traveling Salesman Problem of Second Order* (2-TSP).

2-TSP is also $\mathcal{NP}$-hard, which can be seen as follows. Let c be the weight function of an ATSP instance. If you define the three-dimensional weight function $c'(u, v, w) := c(v, w)\ \forall u \in V \setminus \{v, w\}$, then solving this ATSP instance and the corresponding 2-TSP instance are equivalent. Thus ATSP can be reduced in polynomial time to 2-TSP.

In some sense, 2-TSP and ATSP have the same difficulty, as both have $(n-1)!$ feasible tours. In another sense, 2-TSP is much more difficult, as it has a three-

dimensional weight function and the ATSP only has a two-dimensional one. As we were not able to find an effective polynomial reduction from 2-TSP to ATSP, 2-TSP has to be considered as a new combinatorial optimization problem. The purpose of this paper is to develop heuristics and exact algorithms for this problem and to do an experimental study of these algorithms for random and real instances.

The paper is organized as follows. In Section 2 we propose different heuristics and in Section 3 different exact algorithms for the 2-TSP. In Section 4 we give an experimental study for the heuristics and the exact algorithms. Finally we summarize this paper and give suggestions for future research in Section 5.

2 Heuristics for the Traveling Salesman Problem of Second Order

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.

2.1 Cheapest-Insert Algorithm

The *Cheapest-Insert Algorithm* (CI) is a generalization of an algorithm for the ATSP [23]. We start with an arc (v_1, v_2) as a subtour and choose this arc in such a way that the sum of the costs of a cost minimum predecessor of v_1 and a cost minimum successor of v_2 is minimum. Then step by step, a new vertex is included in the subtour, so that the new subtour is cost minimum. If the tour is complete, we stop this procedure.

2.2 Nearest-Neighbor Algorithm

The *Nearest-Neighbor Algorithm* (NN) is also a generalization of an algorithm for the ATSP [23]. 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 $v_1 \rightarrow v_2$ in such a way that the new path becomes cost minimum. We stop, until 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 in this step only one possibility for the predecessor of v_1 exists, the costs of $(p(p(p(v_1))), p(p(v_1)), p(v_1)))$, $(p(p(v_1)), p(v_1), v_1))$, and $(p(v_1), v_1, s(v_1))$ are irrelevant for this choice. Thus we choose the arc (v_1, v_2) in the first step in such a way that the sum of the costs of a cost minimum successor of v_2 and the average costs of a predecessor of v_1 is cost minimum.

2.3 Two-Directional-Nearest-Neighbor Algorithm

In this section we suggest 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 criteria for this choice is to use the minimum 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 minimum neighbor vertex x and $c(y, v_1, v_2)$ for a cost minimum neighbor vertex y itself 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 minimum neighbor vertex (for an overview over the theory of tolerances see [12, 13]). A similar idea was used for a tolerance based version [11] of the greedy heuristic [7] for the ATSP and a tolerance based version [14] of the contract-or-patch heuristic for the ATSP [7, 16]. Thus we choose the direction from which the upper tolerance value is larger, as not using the cost minimum neighbor vertex would cause a larger jump of the costs of the current path.

2.4 Assignment-Patching Algorithm

A well-known technique for the ATSP is the patching technique, which starts from k cycles, where each vertex is visited exactly by one of the cycles, and then – step by step – patches two cycles together, until there is only one cycle, which is the ATSP tour of this heuristic. The most popular version of ATSP patching is based on the Assignment Problem (AP), which is defined as follows. Let a matrix $C = (c_{ij})_{1 \leq i,j \leq n} \in \mathbb{R}^{n,n}$ be given. Then the AP is to find a node permutation π^* so that $\pi^* = \arg\min \left\{ \sum_{i=1}^{n} c_{i,\pi(i)} : \pi \in \Pi_n \right\}$, where Π_n is the set of all permutations of $\{1, \ldots, n\}$. There are many efficient algorithms for the AP [2, 10, 19] (for an experimental comparison of AP algorithms see [4]). The most efficient one is the Hungarian algorithm, which is based on König-Egervary's theorem and has a complexity of $O(n^3)$. In our algorithm we use the implementation of the Hungarian algorithm by Jonker and Volgenant [27].

The corresponding AP instance to an ATSP instance uses the same weight function c with $c_{ii} = \infty$ for $1 \leq i \leq n$. As the AP solution can be computed efficiently and the solution value is a good lower bound for an optimum ATSP solution value, the AP solution is a good starting point for patching. Karp and Steele suggested for each step to patch the two cycles containing the most number of vertices [21]. 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 minimum 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 minimum. For the 2-TSP the following term has to be minimized:

$$\begin{aligned} &c(p(v_1), v_1, w_2) + c(v_1, w_2, s(w_2)) + c(p(v_2), v_2, w_1) + c(v_2, w_1, s(w_1)) \\ &-c(p(v_1), v_1, w_1) - c(v_1, w_1, s(w_1)) - c(p(v_2), v_2, w_2) - c(v_2, w_2, s(w_2)) \end{aligned} \tag{1}$$

As natural extension of the AP we define for a directed graph $G = (V, E)$ with $n \geq 3$ and a weight 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 $v = w$ the *Assignment Problem of Second Order* (2-AP) as the problem of finding an one-to-one-mapping $f : V \to V$ so that the costs $\sum_{i=1}^{n} c(v_i, f(v_i), f(f(v_i)))$ are minimum.

Recently Fischer and Lau [6] have shown by a reduction from SAT that 2-AP is $\mathcal{NP}$-hard. One way to solve it, is by integer programming (see Section 3.2), which is not fast enough for an efficient heuristic. Instead we suggest to approximate an optimum solution for 2-AP by a polynomial time solvable heuristic solution. For this purpose define a two-dimensional weight function $c' : V \times V \to \mathbb{R}$, which depends on the three-dimensional weight function $c : V \times V \times V \to \mathbb{R}$ as follows: $c'(v, w) = \min_{u \in V \setminus \{v,w\}} c(u, v, w) \quad \text{for } v \neq w \in V$.

The solution of the AP for this weight function, which can be computed in $O(n^3)$, is a lower bound for the 2-AP solution, which we call approximated 2-AP solution. We then patch the cycles of this AP solution, and call the approach *Assignment Patching Algorithm* (AK), where "K" stands for "Karp Steele".

2.5 Nearest-Neighbor-Patching Algorithm

One drawback of the NN algorithm is that the number of remaining vertices becomes smaller (by 1) at each step. Thus in average the difference between the weight 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 exactly, we change the path $(v_1, \ldots, v_i)$ to a cycle, if the sum of the two weights $c(v_{i-1}, v_i, v_1)$ 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 holds $3 \leq i \leq n-3$. We repeat these steps with the remaining vertices, until each vertex is contained in exactly one cycle.

2.6 Two Directional Nearest-Neighbor-Patching Algorithm

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

2.7 Greedy Algorithm

The *Greedy Algorithm* (G) is also a generalization of an ATSP algorithm [7] which is based on the contraction procedure.

Let $G = (V, E)$ be a complete graph with $n \geq 3$ vertices and $c : E \to \mathbb{R}$ a weight 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 weight function $c' : E' \to \mathbb{R}$ defined by

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

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

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

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.

2.8 *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 introduce 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 [22] we need the following definition. Let a complete graph $G = (V, E)$, $|V| = n$ and a complete tour T be given, and let $k \leq n$. Furthermore let a (two-dimensional or three-dimensional) weight 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 after the change is still a tour. T is called *k-optimum*, if no r-OPT step with $r \leq k$ reduces the weight of the tour. Note that in general a k-optimum tour is not unique.

Each tour received by one of the previous construction heuristics can be transformed to a k-optimum tour by doing tour improving k-OPT steps as long, as they exist. As is customary in literature [17] we consider only the case $k \leq 5$.

3 Exact Algorithms for the Traveling Salesman Problem of Second Order

3.1 Branch-and-Bound Algorithm

The following Branch-and-Bound Algorithm (BnB) in the worst case visits all possible tours in a lexicographic order and computes the tour with minimum costs. To avoid that in fact all tours have to be visited, it computes (local) lower bounds and upper bounds by visiting and analyzing subpaths of all possible tours.

First we start with an arbitrary heuristic for the 2-TSP to compute a good upper bound. If at some state of the algorithm we consider a subpath, we compute a lower bound lb for a 2-TSP solution containing this subpath. As lower bound the approximated 2-AP solution (see Section 2.4) is computed. If lb is larger 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 maximum. This choice is used, because we expect more prunes to appear, as the lower bounds in the first steps should be rather large.

3.2 Integer-Programming Algorithm

The AP can be described as an integer program as follows:

$$\min \sum_{i=1}^{n} \sum_{j=1, j\neq i}^{n} c_{ij} x_{ij} \tag{2}$$

$$\sum_{j=1, j\neq i}^{n} x_{ij} \;=\; 1 \;\; \forall 1 \leq i \leq n \tag{3}$$

$$\sum_{i=1, i\neq j}^{n} x_{ij} \;=\; 1 \;\; \forall 1 \leq j \leq n \tag{4}$$

$$x_{ij} \in \{0; 1\} \; \forall 1 \leq i \neq j \leq n \tag{5}$$

where $C = (c_{ij})_{1 \leq i \neq j \leq n}$ is the weight matrix of the AP instance. Equation (3) means that each vertex has exactly one out-arc and equation (4) that each vertex has exactly one in-arc. The AP solution consists of all arcs with $x_{ij} = 1$.

Similarly, we suggest to model the 2-AP by the following integer program:

$$\min \sum_{i=1}^{n} \sum_{j=1,j\neq i}^{n} \sum_{k=1,k\neq i,k\neq j}^{n} c_{ijk} x_{ijk} \tag{6}$$

$$\sum_{j=1,j\neq i}^{n} \sum_{k=1,k\neq i,k\neq j}^{n} x_{ijk} = 1 \ \forall 1 \leq i \leq n \tag{7}$$

$$\sum_{i=1,i\neq j}^{n} \sum_{k=1,k\neq i,k\neq j}^{n} x_{ijk} = 1 \ \forall 1 \leq j \leq n \tag{8}$$

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

$$\sum_{k=1,k\neq i,k\neq j}^{n} x_{ijk} = \sum_{k=1,k\neq i,k\neq j}^{n} x_{kij} \ \forall 1 \leq i \neq j \leq n \tag{10}$$

$$x_{ijk} \in \{0;1\} \ \forall 1 \leq i,j,k \leq n, \ i \neq j, i \neq k, j \neq k \tag{11}$$

where $C = (c_{ijk})_{1\leq i,j,k\leq n,\ i\neq j,i\neq k,j\neq k}$ is the weight matrix of the 2-AP instance.

Equation (7) means that each vertex appears exactly once as a first vertex of a path of three vertices of the tour, equation (8) that each vertex appears exactly once as a second vertex of a path of three vertices of the tour and equation (9) that each vertex appears exactly once as a third vertex of a path of three vertices of the tour. Equation (10) expresses the condition that, if i is a second vertex and j a third vertex of a path of three vertices of the tour, then there is another path of three vertices of the tour, where i is the first vertex and j is the second vertex.

Like for the AP and the ATSP, a solution of the 2-AP consists of $k \geq 1$ cycles. If $k = 1$, the solution of the 2-AP is an optimum 2-TSP solution. To avoid the possibility $k > 1$, we prevent each cycle $(v_{s_1}, \ldots, v_{s_t})$ by adding the following inequality to the integer program:

$$x_{s_{t-1},s_t,s_1} + x_{s_t,s_1,s_2} + \sum_{i=1}^{t-2} x_{s_i,s_{i+1},s_{i+2}} \leq t-1 \tag{12}$$

Unfortunately an exponential number of inequalities of that type exists. For this purpose, we solve the integer program and – step by step – add inequalities of that type which are violated.

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

4 Experimental Study

We implemented all algorithms in C++, where as subroutines we used the AP solver implemented by Jonker and Volgenant [27] and the IP solver Cplex [26].

All experiments were carried out on a PC with an Athlon MP 1900+ CPU with 2GB memory.

As test instances we chose a class of random instances, where each entry c_{ijk} is independently chosen as an integer from $[0, \ldots, 10000]$. For the tests regarding the heuristics we computed the average over 1,000 instances for each dimension $3, 4, \ldots, 44$, and regarding the exact algorithms the average over 10 instances for each dimension $3, 4, \ldots, 24$. Furthermore we considered three real classes *ML*, *BMA*, *MAP* [15], which for each class and each dimension consist of one instance. For the heuristics we used the instances with dimension $3, 4, \ldots, 41$ and for the exact algorithms the instances with dimension $3, 4, \ldots, 17$. We observe that the real instances are much harder to solve than the random instances.

4.1 Comparison of Heuristics

An experimental study of heuristics for the ATSP is given in [18]. In this section we make a similar study for the 2-TSP. 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-OPT Algorithm.

All construction algorithms are rather fast, as they have complexity not worse than $O(n^3)$. We observe that the running times of all seven basic algorithms are very similar (except G which is slower) and the running times of all seven basic algorithms plus OPT steps are also very similar. Comparing the algorithms with and without OPT steps, the algorithms with OPT steps are considerably slower. Thus we only compare the upper bounds, where for the random instances we use the average value over 1,000 instances. We use the versions 2NN and 2NN+OPT as a basis for all comparisons and compare the remaining six algorithms against these versions by the difference of upper bounds. The results for the basic versions can be found in Figure 1 and the results for the OPT versions in Figure 2. If some lines do not appear in the diagrams, it means they have y-value 0. To compare the basic and the OPT versions, we only consider 2NN and 2NN+OPT as a representative. These results are shown in Figure 3.

The experiments show that the OPT versions clearly beat the basic versions. Furthermore the results for the random instances are completely different to the results for the real instances. For the real instances, CI and AK are the best algorithms, whereas for the random instances, 2NN and 2NNK are the best algorithms. AK is the worst algorithm for the random instances, and G is rather bad for all instances. Similar results as for the basic versions hold for the OPT versions.

Finally we compare the upper bounds with the optima computed by one of the exact algorithms BnB or IP. Again the results differ for the real and the random instances. Considering real instances, it holds that for 44 of 45 instances at least one (of seven) OPT heuristics finds the optimum. For the

Fig. 1. Quality comparison of basic heuristics with respect to 2NN

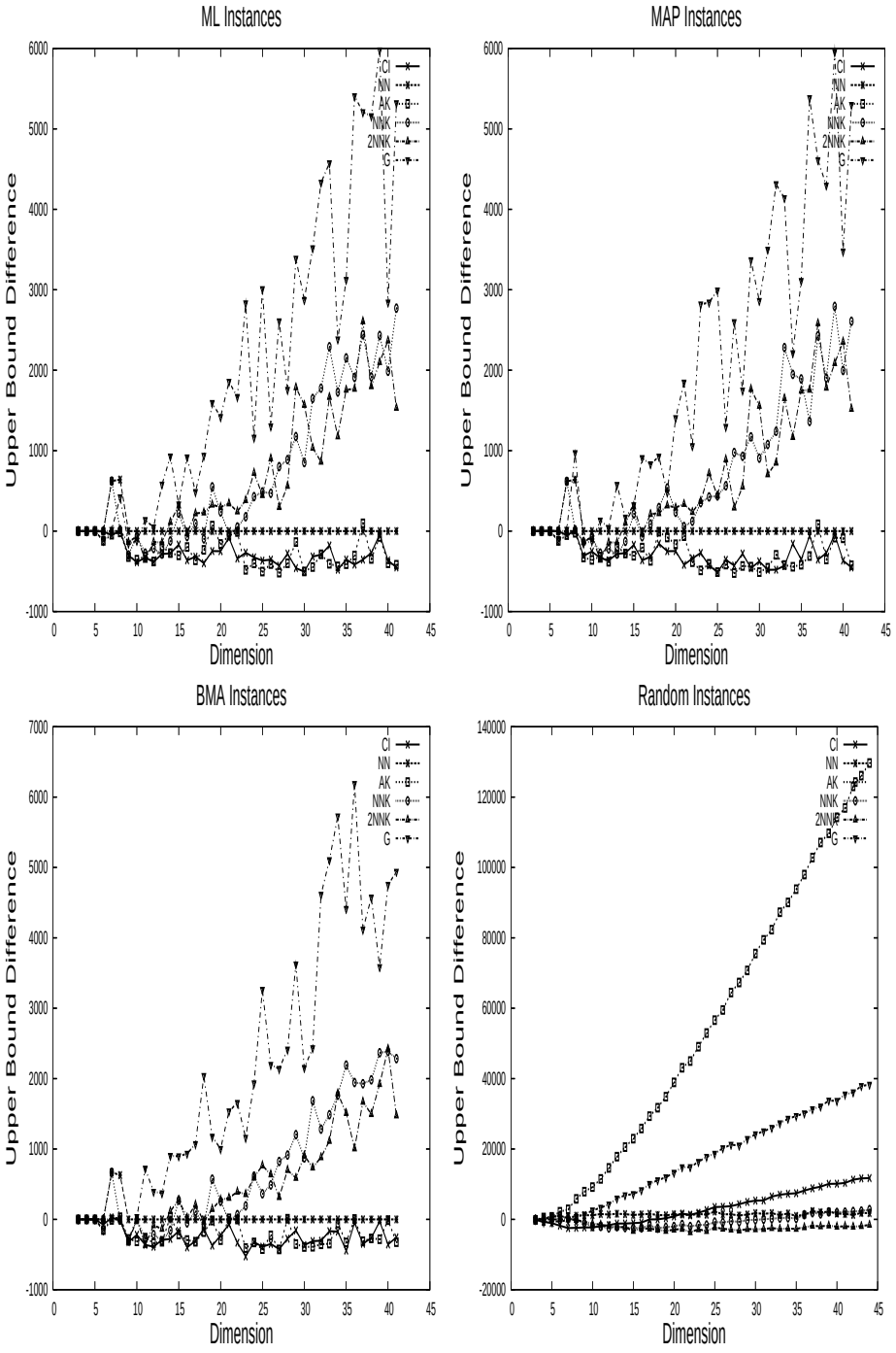


Fig. 2. Quality comparison of OPT heuristics with respect to 2NN+OPT

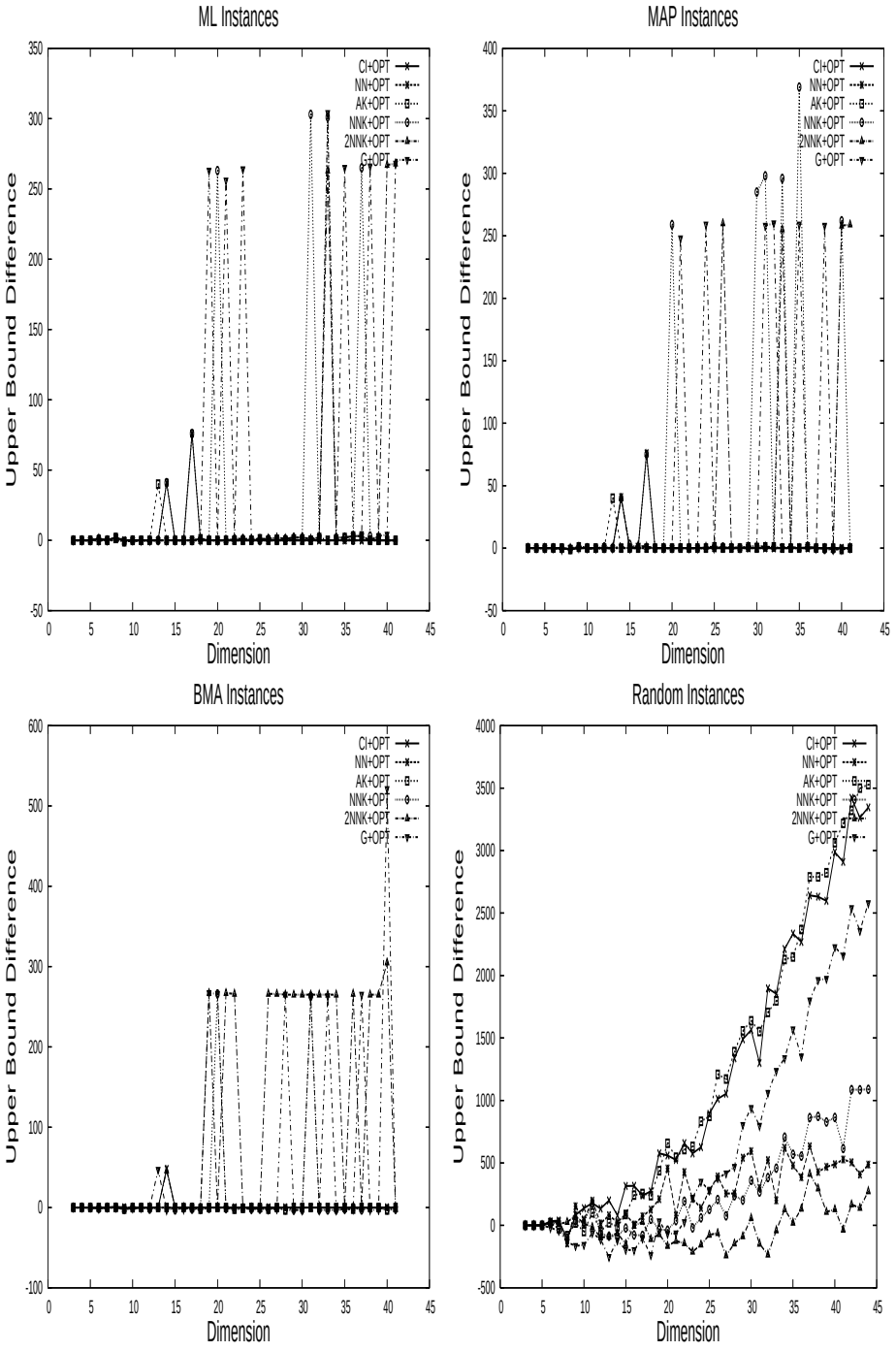


Fig. 3. Quality comparison of 2NN with respect to 2NN+OPT

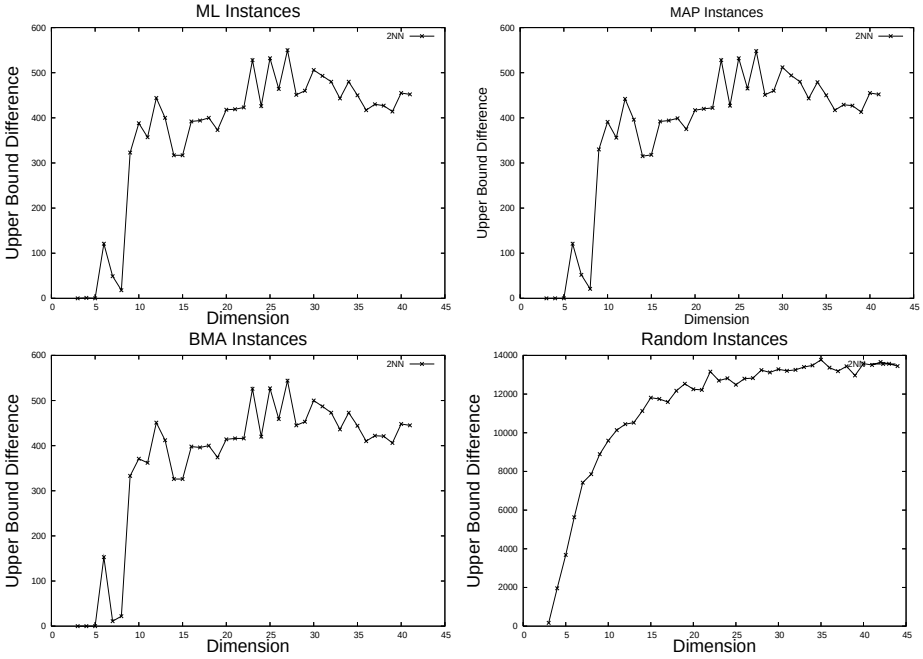
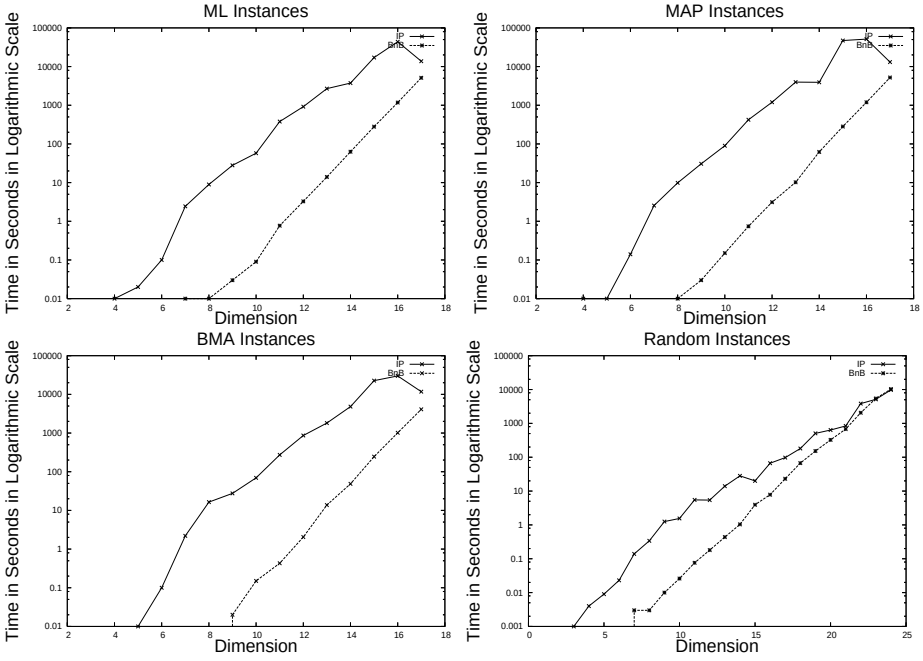


Fig. 4. Running time comparison of IP and BnB



single remaining instance as well as for the unsuccessful heuristics we receive upper bounds very close to the optimum. Note that these results lead to many y-values 0 or at least small y-values in the diagrams. In contrast, for the random instances the upper bounds are more far away from the optimum.

4.2 Comparison of Exact Algorithms

In this section we compare the running times of the IP Algorithm and the BnB Algorithm, where all running times are given in seconds. The results can be found in Figure 4.

We observe that the random instances can be solved much faster than the real instances. Furthermore the BnB Algorithm is more efficient than the IP-Algorithm for the real *and* the random instances of our benchmark set. But on the other hand further experiments showed, that the IP-Algorithm was able to solve the MAP instance of dimension 21 in about three days and the BMA instance of dimension 26 in about three weeks, whereas the BnB Algorithm was not able to solve these instances. Thus for larger dimensions the IP-Algorithm becomes more efficient. For the IP Algorithm, the number of iterations of adding equalities of type (12) is the most important criterion for the running time. For real instances this number is very large: for dimension larger or equal than 10 we have at least 30 iterations. In contrast, the number of iterations is rather small for random instances: not more than 5 iterations for 195 of the 220 examples of dimensions $3, 4, \ldots, 24$. This is the main reason for the running time difference of the IP Algorithm between real and random instances.

5 Summary and Future Research

The purpose of this paper is to introduce a new combinatorial optimization problem with important applications in bioinformatics. We suggest seven heuristics and two exact algorithms for this problem and compare them in an experimental study. For the real instances, the best of our heuristics finds the optima in almost all cases, whereas an exact algorithm is able to compute instances up to dimension 26. For the exact algorithms the running times are considerably smaller for the random instances than for the real instances.

Nevertheless the paper aims to be only a starting point for research in this challenging area. It seems interesting to know, whether completely different heuristics as genetic algorithms [9] or tabu search [8] can be applied for this problem. Also for exact algorithms improvements seem to be possible. For example, the BnB Algorithm might be improved by different branching criteria or better lower bounds. Another possibility is to apply a branch-and-cut approach, as successfully done by the program package Concorde [1, 25] for the STSP. Furthermore integer programming can be improved by cut-and-solve, as shown in [3] for the special case of the ATSP.

Acknowledgement

This work was supported by German Research Foundation (DFG) under grant number MO 645/7-3.

References

1. D.L Applegate, R.E. Bixby, V. Chvátal, W.J. Cook: The Traveling Salesman Problem. A Computational Study. *Princeton University Press*, 2006.
2. D.P. Bertsekas: A New Algorithm for the Assignment Problem. *Math. Program.* **21**, 152-171, 1981.
3. S. Climer, W. Zhang: Cut-and-Solve: An Iterative Search Strategy for Combinatorial Optimization Problems. *Artificial Intelligence* **170(8)**, 714-738, 2006
4. 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.
5. K. Ellrott, C. Yang, F.M. Sladek, T. Jiang: Identifying Transcription Factor Binding Sites Through Markov Chain Optimization. *Bioinformatics* **18**, 100-109, 2002.
6. F. Fischer, A. Lau, University of Chemnitz. Private Communication.
7. F. Glover, G. Gutin, A. Yeo, A. Zverovich: Construction Heuristics for the Asymmetric TSP. *European J. Oper. Res.* **129**, 555-568, 2001.
8. F. Glover, M. Laguna: Tabu Search. *Kluwer*, Dordrecht, 1997.
9. D.E. Goldberg: Genetic Algorithms in Search, Optimization, and Machine Learning. *Addison-Wesley*, Bonn, 1989
10. A.V. Goldberg, R. Kennedy: An Efficient Cost Scaling Algorithm for the Assignment Problem. *Math. Program.* **71**, 153-177, 1995.
11. B. Goldengorin, G. Jäger: How To Make a Greedy Heuristic for the Asymmetric Traveling Salesman Competitive. *SOM Research Report* 05A11, University of Groningen, The Netherlands, 2005.
12. 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.
13. B. Goldengorin, G. Jäger, P. Molitor: Tolerances Applied in Combinatorial Optimization. *J. Comput. Sci.* **2**(9), 716-734, 2006.
14. 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.
15. I. Grosse, University of Halle-Wittenberg, Chair for Bioinformatics. Private Communication.
16. G. Gutin, A. Zverovich: Evaluation of the Contract-or-Patch Heuristic for the Asymmetric TSP. *INFOR* **43**(1), 23-31, 2005.
17. K. Helsgaun: An Effective Implementation of the Lin-Kernighan Traveling Salesman Heuristic. *European J. Oper. Res.* **126**(1), 106-130, 2000.
18. D.S. Johnson, G. Gutin, L.A. McGeoch, A. Yeo, W. Zhang, A. Zverovich. Experimental Analysis of Heuristics for the ATSP. Chapter 10 in: The Traveling Salesman Problem and Its Variations. G. Gutin, A.P. Punnen (Eds.). *Kluwer*, Dordrecht, 445-489, 2002.

19. R. Jonker, A. Volgenant: A Shortest Augmenting Path Algorithm for Dense and Sparse Linear Assignment Problems. *Computing* **38**, 325-340, 1987.
20. 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.
21. 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.
22. S. Lin, B.W. Kernighan: An Effective Heuristic Algorithm for the Traveling-Salesman Problem. *Oper. Res.* **21**, 498-516, 1973.
23. 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.
24. X. Zhao, H. Huang, T.P. Speed. Finding Short DNA Motifs Using Permuted Markov Models. *Journal of Computational Biology* **12**, 894-906, 2005.
25. Source code of [1] (Concorde). Available: "http://www.tsp.gatech.edu/concorde.html".
26. Homepage of CPLEX: "http://www.ilog.com/products/optimization/archive.cfm".
27. Source code of [19]. Available: "http://www.magiclogic.com/assignment.html".