Serdica Journal of Computing 16(1), 2022, pp. 57-70, 10.55630/sjc.2022.16.57-70
Published by Institute of Mathematics and Informatics, Bulgarian Academy of Sciences

Walk on the Hypercube with Minimum Similarities

Georgi Georgiev[1], Nicola Yanev[2], Emil Kelevedjiev[2], Borislav Yurukov[3]

[1]Faculty of Mathematics and Informatics,
Sofia University "St. Kliment Ohridski", Bulgaria
skelet@fmi.uni-sofia.bg

[2]Institute of Mathematics and Informatics,
Bulgarian Academy of Sciences, Bulgaria
choby@math.bas.bg
keleved@math.bas.bg

[3]Department of Informatics,
South-West University "Neofit Rilski", Blagoevgrad, Bulgaria
bobyur@swu.bg

Abstract

An efficient scheduler (algorithm) as a part of batch processing, implemented in a real warehouse management system is considered. The goal is not completion time but rather the *fairness* of the schedule expressed as a minimal overload of working places (machines, workers, etc.) used. As a combinatorial optimization problem, the objective is to find the permutation of the rows of a $n \times k$ boolean matrix B that minimizes the sum of the scalar products of each two consecutive rows. For the above-mentioned warehouse, the size n of a batch is in thousands and the number of working places is up to ten.

The problem is modeled as many visits traveling salesman problem over the vertices of k dimensional unit hypercube with distances equal to the scalar products of the coordinate vectors of the vertices.

The case $n = 2^k$ is proven NP-hard and for the needs of the practice, where $n >> k$ a heuristic greedy algorithm with a good, experimentally proven precision is proposed.

Received: January 31, 2023, *Accepted:* February 27, 2023, *Published:* March 2, 2023
Citation: Georgi Georgiev, Nicola Yanev, Emil Kelevedjiev, Borislav Yurukov, Walk on the Hypercube with Minimum Similarities, Serdica Journal of Computing 16(1), 2022, pp. 57-70, https://doi.org/10.55630/sjc.2022.16.57-70

Keywords: Scheduling, Travelling Salesman Problem, Transportation Problem, Integer Programming

ACM 2012 CCS Concepts: Mathematics of computing → Discrete mathematics → Combinatorics → Combinatorial optimization; Applied computing → Operations research → Transportation

Mathematics Subject Classification 2020: 90B35, 90B06

1 Introduction

Consider the following combinatorial optimization problem:

Find the reordering of the rows of a given $n \times k$ boolean matrix B that minimizes the sum of the scalar products of each two consecutive rows. This is a compact form of a new class of scheduling problems aiming at finding the s.c. friendly schedule (see the definition below). We are given n tasks that need to be scheduled on k unit time parallel machines. Each task j is done in unit time on a subset of $\{1, 2, ..., k\}$, defined by the characteristic vector (row) r_j of B. The scalar product $r_j.r_{j+1}$ gives the number of loaded (active) machines for two consecutive time periods. Thus, if the $i-$th bits of the strings r_j, r_{j+1} are 11 the $i-$th machine is considered overloaded. A schedule will be called friendly if the sum of overloads for each machine is minimal. In bit strings terminology, the fair schedule maximizes the number of 10 substrings or equivalently, minimizes the number of 11 substrings. The difference is in the symmetry: for the min problem both schedules, say prime and reverse are fair, while for the max problem, this could not be the case.

The motivation comes from our involvement in the creation of an algorithm to be used in the optimization module of a computerized system for the management of a huge really existing warehouse. The n tasks are boxes to be filled with the clients' orders and the machines (workers) are k working places along the conveyors used to convey the boxes. Since the completion time depends only on the speed of the conveyor and the number of boxes, the goal is to schedule (order) the batch of boxes (tasks) to reduce the workload of the worker(s) over short periods of time, or in the context of the above-mentioned terminology, as much as possible "work and then rest" phases. If we want to be consistent with the scheduling theory terminology and call $r_j.r_{j+1}$ delay on task $j, j = 1, ..., n-1$, the problem is to find a schedule with minimum makespan.

2 Definitions

The problem we study: In what follows the task will be associated with the set of vertices of k-dimensional hypercube B^k. What was called the delay between tasks α and β is simply the scalar product of the corresponding vertices $d(\alpha, \beta) = \sum_{i=1}^{k} \alpha_i \beta_i$.

A graph-theoretic view: By adding a null task (corresponding to the vertex zero of B^k) we obtain an *augmented* tasks set. Clearly, any ordering of the original task set corresponds bijectively to a circular ordering of the augmented task set.

Let $G = (V, E)$ be the complete undirected graph with edge weights such that V is the augmented set of tasks and the edge weights are the distances. It is easy to see that computing an optimal scheduling is the same as to solving the Traveling Salesman Problem (see [1]) on G.

Let n and k be defined as above and $l = 2^k$.

Let $I_k = \{0, 1, \ldots, l-1\}$ be the set of integer numbers from 0 to $l-1$. This set equals the vector set of B^k.

Given a vertex set $V = \{v_1, v_2, \ldots, v_n\}$ we call *scalar graph* the triple $\langle V, E, f \rangle$, where E is an edge set such that (V, E) forms a complete graph and $f : V \to I_k$ is some function.

The scalar graph has both vertex weights defined by f and edge weights defined by $w(u, v) = d(f(u), f(v))$. For brevity, we denote scalar graphs as $\langle V, f \rangle$.

With $scTSP$ we denote TSP restricted to scalar graphs.

3 Complexity

Lemma 1. *The problem $scTSP$ is* ***NP****-hard.*

Proof. We reduce the Hamiltonian Cycle problem, which is known to be **NP**-complete (see [2]), to $scTSP$. Given an undirected graph $G(V, E)$, let G_1 be a complete graph with vertices V and weight function $w(u, v) = 0$ if $(u, v) \in E$ and $w(u, v) = 1$ if $(u, v) \notin E$. Obviously, the graph G has a Hamiltonian cycle if and only if the optimal solution of the TSP for graph G_1 has weight 0. Let us denote with $e_1, e_2, \ldots, e_m$ the edges in G_1 with weight 1.

Now we build a scalar graph $G_2 = \langle V, f \rangle, f : V \to I_m$ as follows. For any $v \in V$, define $f(v) = \alpha_1 \alpha_2 \ldots \alpha_m$, where $\alpha_i = 1$ if and only if v is incident to edge e_i.

Let for some other vertex $u \in V$ $f(u) = \beta_1\beta_2\ldots\beta_m$. The product $\alpha_i\beta_i$ is 1 only in the case when $e_i = (u, v)$. It follows that the scalar product $(f(u), f(v))$ for graph G_2 is the same as the weight function $w(u, v)$ in G_1.

Thus the graph G has a Hamiltonian cycle exactly when the optimal solution of the TSP for graph G_2 has weight 0. We can construct G_2 in time polynomial in the size of G, which completes the proof. $\square$

scCSP is **NP**-hard too. The construction from Lemma 1 reduces HP (Hamiltonian Path problem) to *scCSP*.

4 A case study – scalar hypercube

Consider the graph of all the vertices of k-dimensional hypercube with scalar distances between the corresponding Boolean strings.

Let us call this graph SD_k. Technically we can treat it as a scalar graph $\langle I_k, I\rangle$, where $I : I_k \to I_k$ is identity. Let $v(scTSP)$ be the optimal objective function value (the minimal length of the traveling salesman tour on SD_k). Then the following is true:

Lemma 2. $v(scTSP) \geq 2^{k-2}$.

Proof. Let $\langle v_1, v_2, \ldots, v_n, v_1\rangle$ be an optimal solution, d_i be a distance between v_i and v_{i+1}, and $e_i = |v_i| + |v_{i+1}| - k$, where $|v|$ is the bits number of vertex v.

The following relations are obvious:

$$e_i \leq d_i \tag{1}$$

$$\sum_1^n e_i = 0 \tag{2}$$

Let E_+, E_0, E_- be the sets of edges in the optimal cycle with positive, zero, or negative value of e_i resp. From (2) follows:

$$\sum_{E_0} e_i = 0 \tag{3}$$

$$\sum_{E_-} e_i + \sum_{E_+} e_i = 0 \tag{4}$$

$$\sum_{E_-} |e_i| = \sum_{E_+} e_i \tag{5}$$

Each sum in (5) is at least the cardinality of the set on which we sum up, therefore:

$$\max\left(|E_-|, |E_+|\right) \leq \sum_{E_-} |e_i| = \sum_{E_+} e_i \tag{6}$$

But $\frac{|E_-|+|E_+|}{2} \leq \max\left(|E_-|, |E_+|\right)$, therefore:

$$\frac{|E_-| + |E_+|}{2} \leq \sum_{E_-} |e_i| = \sum_{E_+} e_i \tag{7}$$

We apply (1) on the elements of E_+ and get:

$$\frac{|E_-| + |E_+|}{2} \leq \sum_{E_+} d_i \tag{8}$$

Let us consider the set E_0. If $|E_0| \leq \frac{n}{2}$, then $|E_-| + |E_+| \geq \frac{n}{2} = 2^{k-1}$ and from (8) directly follows the statement of the lemma:

$$2^{k-2} = \frac{n}{4} \leq \frac{|E_-| + |E_+|}{2} \leq \sum_{E_+} d_i \leq \sum_{1}^{n} d_i \tag{9}$$

In the remaining case $|E_0| > \frac{n}{2}$. Let $|E_0| = \frac{n}{2} + l,\ l > 0$. It is easily seen that at most $\frac{n}{2}$ edges in E_0 have weight 0, and they are edges with opposite Boolean vectors. Hence:

$$l \leq \sum_{E_0} d_i \tag{10}$$

From (10) follows a weaker inequality:

$$\frac{l}{2} \leq \sum_{E_0} d_i \tag{11}$$

We sum up (8) and (11):

$$\frac{l + |E_-| + |E_+|}{2} \leq \sum_{E_0} d_i + \sum_{E_+} d_i \tag{12}$$

But $l + |E_-| + |E_+| = \frac{n}{2} = 2^{k-1}$ and still we get the assertion of the lemma:

$$2^{k-2} = \frac{l + |E_-| + |E_+|}{2} \leq \sum_{E_0} d_i + \sum_{E_+} d_i \leq \sum_{1}^{n} d_i. \tag{13}$$

□

Lemma 3. *The bound 2^{k-2} is sharp.*

Proof. In order to construct a cycle known as the Anti-Gray code (see [3]), the set of vertices of SD_k is partitioned into two parts I_k^0 and I_k^1, containing vectors with leading zero or one resp.

We interpret the set I_k^0 as a set of vertices of a hypercube of dimension $k-1$ and build a Hamiltonian cycle in it (also known as Grey code). With the convention $l = 2^k$, let this cycle has the form $Gr_0 = \langle v_1, v_2, \ldots, v_{l/2}, v_1\rangle$. It is Well known that successive Boolean vectors in this cycle differ in exactly one position. If $v'_i = 1\overline{\alpha_2\alpha_3}\ldots\overline{\alpha_k}$ is put in correspondence to $v_i = 0\alpha_2\alpha_3\ldots\alpha_k$ then obviously, $Gr_1 = \langle v'_1, v'_2, \ldots, v'_{l/2}, v'_1\rangle$ is a Hamiltonian cycle in I_k^1.

Now merge the two cycles in a cycle $\langle v_1, v'_1, v_2, v'_2, \ldots, v_{l/2}, v'_{l/2}, v_1\rangle$.

The resulting cycle has exactly l vertices. The $l/2$ edges $\langle v_i, v'_i\rangle$ have their ends in opposite vectors and such edges have zero weight.

For edges $\langle v'_i, v_{i+1}\rangle$ there are two cases: When v_i has one unit more than v_{i+1}, the weight of the edge will be 0; otherwise, when v_i has one unit less than v_{i+1}, the weight of the edge will be 1.

In Gray code $\langle v_1, v_2, \ldots, v_{l/2}, v_1\rangle$ the number of edges, when the bits increase, match the number of reductions. Therefore, the edges of the kind $\langle v'_i, v_{i+1}\rangle$ with unit weight, will be exactly $l/4$, or 2^{k-2}. □

An example of an anti-Gray code for $k = 3$ (not written the first vertex at the end of the cycle):

First, build a Gray code for Boolean square – $00, 01, 11, 10$ supplementing it with a 0 in front.

We get $Gr_0 = \langle 000, 001, 011, 010\rangle$ and $Gr_1 = \langle 111, 110, 100, 101\rangle$. Merge them and get cycle $\langle 000, 111, 001, 110, 011, 100, 010, 101\rangle$ with weight 2.

5 Relaxation and approximation

5.1 Relaxation to transportation problem

Let $l = 2^k$ be the cardinality of I_k. For a scalar graph $\langle V, f\rangle$, let a_i be the number of vertices $x \in V$, such that $f(x) = i$.

The array $A = (a_0, a_1, \ldots a_{l-1})$ fully defines the graph $\langle V, f\rangle$ and is convenient for the presentation of practical tasks for which k is much smaller than n.

Every Hamiltonian cycle in $\langle V, f\rangle$ corresponds to an Eulerian multigraph, with vertices in I_k, so that for each vertex i, exactly a_i edges entering it and leaving it [4].

The $scTSP$ is reduced to finding the shortest Eulerian multigraph with a vertex set I_k, and such that for each vertex i the indegree $d^-(i)$ equals the outdegree$d^+(i)$ and $d^-(i) = d^+(i) = a_i$.

If the connectivity condition (for the Eulerian multigraph) is relaxed and denote by x_{ij} the number of edges from i to vertex j, and by c_{ij} the scalar product of the binary presentations of i and j, we get the following linear model:

$$\min \sum_{i,j \in I_k} c_{ij} x_{ij} \tag{14}$$

subject to constraints:

$$\begin{aligned} \sum_{j \in I_k} x_{ij} = a_i,\ i \in I_k,\ x_{ij} \geq 0 \\ \sum_{i \in I_k} x_{ij} = a_j,\ j \in I_k,\ x_{ij} \geq 0 \end{aligned} \tag{15}$$

The model defines an instance of the transportation problem. This problem is efficiently solvable by many existing polynomial algorithms, but its solution is not necessarily transposable to an Eulerian walk. Similar relaxations are considered in [5, p. 540] and [4].

To disable disconnectedness due to the subcycles, we need to add extra constraints like:

$$\sum_{i \in T, j \in I_k/T} x_{ij} \geq 1, \forall T \subsetneq I_k \tag{16}$$

The number of constraints (16) is exponential and the matrix of constraints is not unimodular, i.e. we should require x_{ij} to be integer.

The model given by (14), (15) and (16) will solve $scTSP$ using an algorithm for integer linear programming, but the running time is unpredictable even for small tasks.

Assuming $\mathbf{P} \neq \mathbf{NP}$, any algorithm for exact solving of $scTSP$ would be unacceptable in terms of time complexity.

5.2 Greedy approximation

To solve practical problems, it is better to have an efficient algorithm, which may give an approximate solution. The optimal solution x_{ij} of the transportation problem (14) – (15) is a good start for a greedy approach. It defines multigraph $X = \{\langle i, j, x_{ij} \rangle |\ i, j \in I_k,\ x_{ij} > 0\}$, set by multi-edges of the kind

Algorithm 5.1 Component connection step

```
1: procedure $JoinGreedy(X)$                ▷ The multigraph $X$ satisfies (15)
2:     Find edges $e_1$ and $e_2$ from distinct connected components,
       minimizing $\Delta = min(\delta_+, \delta_-)$
3:     if $\Delta$ is reached by $\delta_+$ then
4:         Glue forward to edges $e_1$ and $e_2$
5:     else
6:         Glue backward to edges $e_1$ and $e_2$
7:     end if
8: end procedure
```

$\langle i, j, x_{ij} \rangle$, where x_{ij} is the number of parallel edges from vertex i to vertex j. We denote by Z_X the cost of X, $Z_X = \sum_{i,j \in I_k} c_{ij} x_{ij}$.

Provided that X has at least 2 connected components (otherwise x_{ij} could be transposed into an optimal Eulerian cycle), the main greedy step – gluing is as follows: Let the multi-edges $e_1 = \langle i_1, j_1, c_1 \rangle$ and $e_2 = \langle i_2, j_2, c_2 \rangle$ belong to different components C_1 and C_2.

Forward gluing of the edges e_1 and e_2 is the replacement of the edges $\langle i_1, j_1 \rangle$ and $\langle i_2, j_2 \rangle$ with two new edges $\langle i_1, j_2 \rangle$ and $\langle i_2, j_1 \rangle$. This replacement preserves the feasibility of the transportation problem from above increasing transportation costs by $\delta_+ = c_{i_1 j_2} + c_{i_2 j_1} - c_{i_1 j_1} - c_{i_2 j_2}$ and adding at most two multi-edges to the corresponding multigraph X' .

Backward gluing of the edges e_1 and e_2 is the reversing of the direction in all edges in C_2, followed by the replacement of the edges $\langle i_1, j_1 \rangle$ and $\langle i_2, j_2 \rangle$ with two new edges $\langle i_1, i_2 \rangle$ and $\langle j_2, j_1 \rangle$. This is again a feasible solution increasing the transportation costs by $\delta_- = c_{i_1 i_2} + c_{j_2 j_1} - c_{i_1 j_1} - c_{i_2 j_2}$. As in *Forward gluing*, the number of edges in the multigraph X' is increased by at most two.

Both kinds of gluing connect component C_1 and C_2 in a new component C', which is an Eulerian subgraph of X'.

The greedy step of our approximation for $scTSP$ is Algorithm 5.1.

Let us use an adjacency list representation for multigraph X with vertices V and multiedges E and simple mapping of the form $\langle vertex, component \rangle$.

The time complexity of step 2 in *JoinGreedy* is $\Theta(|E|^2)$. The complexity of step 4 is $\Theta(|V|)$, needed to remap merging components. The complexity of step 6 is $\Theta(|V| + |E|)$ (additional time of $\Theta(|V| + |E|)$ to reverse edges in C_2). Thus, the complexity of *JoinGreedy* is $\Theta(|V| + |E|^2)$.

The main approximation for $scTSP$ is Algorithm 5.2.

Algorithm 5.2 Greedy solver for *scTSP*

```
1: procedure scGreedy($A[0 \dots l-1]$)                    ▷ $A = (a_0, a_1, \dots, a_{l-1})$
2:     $X \leftarrow$ solution of the transportation problem, defined by $A$
3:     Evaluate connected components in $X$
4:     while $X$ is not connected do
5:         $JoinGreedy(X)$
6:     end while
7:     return Eulerian cycle in $X$
8: end procedure
```

Assuming the dimension of the matrix (c_{ij}) is $l \times l$ and $\sum a_i = n$, let us denote time complexity of step 2 in *scGreedy* with $T_0(l, n)$. After step 2 number of vertices in X is at most l and the number of multi-edges is at most $2l - 1$ (the size of the basis of the transportation problem we solve).

Step 3 is computed by algorithm BFS for time $\Theta(l)$. Step 4 is executed at most l times, each call to $JoinGreedy(X)$ will add at most 2 edges to X. At any time the number of multi-edges in X will be less than $4l$. This gives total complexity $O(l^3)$ for the connectivity phase (steps 3 and 4).

Step 7 has time complexity $\Theta(n)$, giving total complexity $T_0(l, n) + O(n + l^3)$ for *scGreedy*.

The step $JoinGreedy(X)$ is the most trivial way for gluing components in such a relaxation. In the general case of TSP it can generate a randomly large increase in the objective function and deviation from the optimal solution. In the special case of *scTSP* this step works well.

Explanation. The reasons for such behavior are not entirely clear. One possible explanation is by means of geometric interpretation. If we consider the vertices as points of k-dimensional sphere, the cycles in the optimal solutions (of the transportation problem and *scTSP*) alternate - each edge aims at being close to the diameter of the sphere. We can liken each connected component in the solution of TP to a beam of edges, oriented around some diameter. $JoinGreedy(X)$ actually finds two strands with similar orientations and stitches them together resulting in a larger bundle, but the cost of gluing is limited.

Example. The scalar hypercube SD_k has 2^k vertices and transportation problem for it creates 2^{k-1} cycles with cost 0 (each vertex makes a cycle with its opposite Boolean vector). The optimal solution of *scTSP* has cost 2^{k-2}. *scGreedy* solves it for $k < 7$ exactly, but for $k = 7$ the error is 1, and for $k = 8$ the error is 2.

5.3 Improvements

When the array A is sparse, we compact it as follows: Let A contains q nonzero elements, $q \ll l$.

Let $P = (p_1, p_2 \dots p_q), B = (b_1, b_2 \dots b_q)$, where P is an array of all indices of nonzero members of A and B contains corresponding values $B[i] = A[P[i]], 1 \leq i \leq q$.

Tuple $\langle P, B \rangle$ is a dense record for A.

The compact representation $\langle P, N \rangle$ is a natural for $scCSP$. Through the two arrays, we indicate which Boolean masks are used and how many tasks correspond to each of these masks.

If we rewrite programs *scGreedy* to use $\langle P, B \rangle$ as an input, the resulting program *scGreedyC* use P only in two steps:

In step 2 P is used to generate a compact version for transportation problem matrix $(c'_{ij} = (P[i], P[j]))$ with size $q \times q$.

In step 7 P is used to output the Eulerian cycle.

Muligraph X in this implementation has q vertices, less than $4q$ multi-edges and time complexity of *scGreedyC* is $T_0(q, n) + O(n + q^3)$.

If we use binary heap to store the triples $\langle \Delta, e_1, e_2 \rangle$ for all possible gluing in the multigraph X, we can modify *scGreedyC* so that the step of gluing to have complexity $O(q^2 \lg q)$.

5.4 Exact solving for small k and a hypothesis

For solving the transportation problem we used the open-source program lp_solve [6]. Improvements for *scGreedy*, described in section 5.3 are not implemented.

To conduct experiments we developed a computational Algorithm 5.3 for exact solving of $scTSP$.

It is interesting that an unspecified part of the algorithms is step 4. Its aim is to prohibit the appearance of several connected components in X. In all cases, the matrix of constraints is not totally unimodular and we must add the requirement to the lp_solve for the variables to be integers.

Initially, we used constraints of the kind (16) by components obtained in previous unconnected solutions for X. They are long inequalities and lp_solve is working hours even on examples in which $k = 5$.

Better results give a constraint of the type:

$$\sum_{i,j \in C} x_{ij} < \sum_{i \in C} a_i \tag{17}$$

Algorithm 5.3 Exact solver for *scTSP*

```
1: procedure $scTSPsolver(A[0 \dots l-1])$        ▷ $A = (a_0, a_1, \dots, a_{l-1})$
2:     $X \leftarrow$ solution of the transportation problem, defined by $A$
3:     while $X$ is not connected do
4:         Add new constraints to the LP model
5:         $X \leftarrow$ lp_solve generated solution of the LP model
6:     end while
7:     return Eulerian cycle in $X$
8: end procedure
```

where C is a connected component in X, and the sum on the left side is taken only from the non-zero variables in the solution of the model that gave rise to the component. These constraints are shorter and the simplex method provides a solution with a small number of non-integer variables. Using restrictions (17) gives a quick solving (no more than within a minute) for tasks of size $k \leq 6$.

For any scalar graph $\langle V, f \rangle$ we denote by Z_{TP} the cost $\sum\limits_{i,j \in I_k} c_{ij}x_{ij}$ of the optimal solution of the transportation problem, and by Z_{TSP} the cost of optimal solution of *scTSP*, also by Z_{Greedy} the cost of a solution, obtained by the approximation algorithm.

There is an obvious chain of inequalities: $Z_{TP} \leq Z_{TSP} \leq Z_{Greedy}$.

A large number of computational experiments give us reason to say that the following conjecture is likely true:

Conjecture 1. *Let $\langle V, f \rangle$ be a scalar graph, represented by the matrix $A = (a_0, a_1, \dots a_{l-1})$, $l = 2^k$.*

For $\langle V, f \rangle$ the inequality $Z_{TSP} - Z_{TP} \leq \frac{l}{4} = 2^{k-2}$ holds.

Equality is attained only when:

(a) For the elements of A is fulfilled the condition $\forall i,\ a_i = a_{\bar{i}} > 0$, where $\bar{i}$ is obtained by reversing the bits of i ($\bar{i} = 2^k - 1 - i$).

(b) The multigraph X, defined by any solution to the transportation problem for $\langle V, f \rangle$ has exactly $\frac{l}{2}$ connected components.

Statements (a) and (b) are equivalent.

For the scalar hypercube SD_k, all the elements in A are equal to 1. The hypothesis states that on fixed k the graph SD_k is the worst case in terms of the relaxation of *scTSP* to the transportation problem.

k	$l =$ 2^k	a $_{max}$	P $_{sparse}$	$tests$	ave (Z_{ST})	max (Z_{ST})	ave (Z_{GS})	max (Z_{GS})	T $_{Greedy}$	T $_{TSP}$
4	16	1	0.20	1000	1.584	4	0.114	1	0.024	0.12
4	16	4	0.20	1000	0.266	2	0.105	1	0.026	0.054
4	16	1	0.50	1000	0.673	4	0.05	1	0.021	0.066
4	16	4	0.50	1000	0.129	2	0.026	1	0.021	0.042
4	16	9	0.50	1000	0.056	1	0.007	1	0.025	0.029
4	16	1	0.80	1000	0.181	3	0.004	1	0.019	0.038
4	16	4	0.80	1000	0.02	1	0	0	0.016	0.03
5	32	1	0.20	500	3.172	8	0.354	2	0.082	0.702
5	32	4	0.20	500	0.5	3	0.348	2	0.086	0.28
5	32	1	0.50	500	1.258	5	0.258	2	0.08	0.354
5	32	4	0.50	500	0.344	2	0.136	2	0.058	0.196
5	32	9	0.50	500	0.144	3	0.048	1	0.07	0.136
5	32	1	0.80	500	0.514	3	0.03	1	0.054	0.168
5	32	4	0.80	500	0.11	2	0.01	1	0.068	0.098

Table 1: Numerical experiments for small sizes of the problem ($k = 4, 5$).

6 Computational experiments

For small values of k were generated and solved groups of tasks. In every generation with probability P_{sparse} coefficients a_i in A are equal to zero, and non-zero coefficients accept a random value between 1 and a_{max}.

In tables 1, 2, and 3, at fixed k, a_{max} and P_{sparse} are generated and solved *tests* tasks.

We denote by Z_{ST} the difference $Z_{TSP} - Z_{TP}$ for each generated task. Accordingly, $Z_{GS} = Z_{Greedy} - Z_{TSP}$ and $Z_{GT} = Z_{Greedy} - Z_{TP}$.

Z_{GS} is the difference between the cost of the solution found by *scGreedy* and the optimal solution. Obvious relations are $Z_{GT} = Z_{GS} + Z_{GT}$ and $Z_{GS} \leq Z_{GT}$.

T_{Greedy} is the average running time in seconds for the work of the approximate algorithm, and T_{TSP} is the average running time of the exact algorithm.

At $k > 5$, in search of the exact solution in some cases the software package *lp_solve* finds no integer solution of the linear model, or it finds very slow. The reason for such behavior is not entirely clear, but in reducing the depth of branching (starting with the option "-depth 25" instead of the default value 50) loops do not appear. Since that correction can cause skipping optimum, in the Table 2 we present by T^*_{TSP} the average time to find the exact solution.

In tables 1 and 2, we present by $ave(X_{ST})$ the means average of $Z_{TSP} - Z_{TP}$,

k	$l = 2^k$	a_{max}	P_{sparse}	*tests*	ave (Z_{ST})	max (Z_{ST})	ave (Z_{GS})	max (Z_{GS})	T $_{Greedy}$	T $_{TSP}$
6	64	1	0.20	100	6.99	13	0.53	2	0.6	9.8
6	64	4	0.20	100	1.12	4	0.82	3	0.53	4.77
6	64	1	0.50	100	2.35	6	0.98	3	0.47	3.76
6	64	4	0.50	100	0.68	3	0.3	2	0.38	2.53
6	64	9	0.50	100	0.31	2	0.12	2	0.36	2.57
6	64	1	0.80	100	0.97	4	0.2	1	0.35	1.37
6	64	4	0.80	100	0.18	1	0.05	2	0.37	0.78

Table 2: Numerical experiments for $k = 6$.

k	$l = 2^k$	a_{max}	P_{sparse}	*tests*	ave(Z_{GT})	max(Z_{GT})	T$_{Greedy}$
7	128	1	0.20	100	16.31	22	6.36
7	128	4	0.20	100	4.29	8	5.91
7	128	1	0.50	100	7.06	11	4.76
7	128	4	0.50	100	2.37	7	4.61
7	128	9	0.50	100	1	3	4.66
7	128	1	0.80	100	2.78	6	2.87
7	128	4	0.80	100	0.74	3	3.21
8	256	1	0.20	100	33.92	41	81.3
8	256	4	0.20	100	10.19	15	72.12
8	256	1	0.50	100	16.23	22	59.43
8	256	4	0.50	100	5.29	11	56.34
8	256	9	0.50	100	2.71	8	53.92
8	256	1	0.80	100	5.93	10	38.27
8	256	4	0.80	100	1.79	5	40.91
9	512	1	0.20	20	71.05	78	1204.45
9	512	4	0.20	20	23.45	28	1046.75
9	512	1	0.50	20	35.55	41	850.5
9	512	4	0.50	20	12.7	17	806.35
9	512	9	0.50	20	4.45	8	777.15
9	512	1	0.80	20	11.6	16	627.35
9	512	4	0.80	20	4.1	7	601.1

Table 3: Numerical experiments for middle size problems ($k = 7, 8, 9$).

for all generated tasks and by $ave(Z_{GS})$ the average inaccuracy of the greedy algorithm. It is seen that the difference between the approximate and the exact solution Z_{GS} for small tasks ($k < 7$) is acceptable in the worst case and small in the average case.

At $k > 6$ the time for finding the exact solution increases unacceptably. In Table 3 we show using only the greedy algorithm. In this case, Z_{GS} cannot be evaluated, and $Z_{GT} = Z_{Greedy} - Z_{TP}$ is too rough an upper limit of inaccuracy.

We hope that in the majority of cases, the inaccuracy Z_{GS} is acceptable for tasks, in which finding the optimal solution is not critical.

At $k = 9$, *scGreedy* algorithm works slowly. If we make improvements in Section 5.3, the same behavior will be observed when the number q of non-zero elements of the matrix A increases over 500.

The reason is that the complexity of *scGreedy* is dominated by the complexity of the transportation problem. To solve tasks quickly for $q > 500$, an approximate algorithm is required, which does not use the solution of the transportation problem.

Another option for an acceleration of *scGreedy* is to break solving the transportation problem before reaching the optimal solution. We will get a faster, but less accurate solution.

References

[1] C. H. Papadimitriou , K. Steiglitz, *Combinatorial Optimization: Algorithms and Complexity*, Prentice Hall, Englewood Cliffs, New Jersey, 1982.

[2] M. R. Garey, D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman and Company, New York, 1979.

[3] M. Kurt, C. Atilgan, M. E. Berberler, "A Dynamic Programming Approach for Generating N-ary Reflected Gray Code List", *Ege University, Journal of the Faculty of Science*, 2013.

[4] S. S. Cosmadakis, C. H. Papadimitriou, "The Traveling Salesman Problem with Many Visits to Few Cities", *SIAM Journal on Computing*, 13(1):99-108, 1984.

[5] R. E. Burkard, V. G. Deineko, R. van Dal, J. A. A. van der Veen, G. J. Woeginger, "Well-Solvable Special Cases of the Traveling Salesman Problem: A Survey", *SIAM Review*, 40(3):496-546, 1998.

[6] M. Berkelaar, K. Eikland, P. Notebaert, et al., "Lp_Solve: Mixed Integer Linear Programming (MILP) solver", Eindhoven University of Technology, 2004.