STATISTICS, OPTIMIZATION AND INFORMATION COMPUTING
Stat., Optim. Inf. Comput., Vol. **12**, September 2024, pp 1425–1459.
Published online in International Academic Press (www.IAPress.org)

Fast approximation of the traveling salesman problem shortest route by rectangular cell clustering pattern to parallelize solving

Vadim Romanuke *

Faculty of Mechanical and Electrical Engineering, Polish Naval Academy, Gdynia, Poland

Abstract A method of quickly obtaining an approximate solution to the traveling salesman problem (TSP) is suggested, where a dramatic computational speedup is guaranteed. The initial TSP is broken into open-loop TSPs by using a clustering method. The clustering method is based on either imposing a rectangular lattice on the nodes or dividing the dataset iteratively until the open-loop TSPs become sufficiently small. The open-loop TSPs are independent and so they can be solved in parallel without synchronization, whichever the solver is. Then the open-loop subroutes are assembled into an approximately shortest route of the initial TSP via the shortest connections. The assemblage pattern is a symmetric rectangular closed-loop serpentine. The iterative clustering can use the rectangular assembling approach as well. Alternatively, the iterative clustering can use the centroid TSP assembling approach, which requires solving a supplementary closed-loop TSP whose nodes are centroids of the open-loop-TSP clusters. Based on results of numerical simulation, it is ascertained that both the iterative clustering and rectangular cell clustering pattern are roughly equally accurate, but the latter is way more computationally efficient on squarish datasets. Fast approximation of the TSP shortest route serves as an upper bound. In addition, the route can be studied to find its bottlenecks, which subsequently are separated and another bunch of open-loop TSPs is approximately solved. For big-sized TSPs, where the balance of the accuracy loss and computational time is uncertain, the rectangular cell clustering pattern allows obtaining fast solution approximations on multitudinous non-synchronized parallel processor cores.

Keywords Traveling salesman problem, Clustered subproblems, Open-loop route, Parallelization, Route length, Genetic algorithm, Rectangular cell clustering, Mona Lisa problem

AMS 2010 subject classifications 90C08, 90C09, 90C27

DOI: 10.19139/soic-2310-5070-2037

1. Traveling salesman problem solution approximation

In combinatorial optimization, the traveling salesman problem (TSP) is often referred to as the most important problem whose solution for any number of nodes would lead to solving a variety of related problems [25, 36]. However, the existing methods of finding routes of the minimal length are limited to either a few hundred nodes or being designed for solving specific TSPs [10, 22]. Moreover, the supplementary task of finding all versions of the TSP shortest route is much harder [36, 1]. Whereas the exact methods are inapplicable to big-sized TSPs not returning a single solution, it is obvious that the tradeoff here is quite acceptable as for balancing the accuracy loss and computational time (resources), as well as for finding at least a subset of the TSP approximate solutions [38, 52]. The tradeoff implies finding approximately shortest routes (whose lengths may differ) rather than the exactly shortest routes (of the same, minimized, length) by dramatically speeding up the process of solving [6, 17]. However, the balance of the accuracy loss and computational time usually faces uncertainty [29, 8]. It is reasoned and established mainly using experience and expert judgments depending on an engineering application [36, 5, 20].

*Correspondence to: Vadim Romanuke (Email: romanukevadimv@gmail.com). Faculty of Mechanical and Electrical Engineering, Polish Naval Academy, 69 Śmidowicza Street, Gdynia, Poland, 81-127.

ISSN 2310-5070 (online) ISSN 2311-004X (print)

One of the most accurate and fastest TSP solution approximations is ensured by genetic algorithms [38, 52, 4, 15]. They converge to an approximate solution of the TSP faster than the algorithms of ant colony optimization [47, 45], simulated annealing [46, 16], and tabu search [9]. Designed for solving almost any combinatorial optimization problem, genetic algorithms do not aim at finding exact solutions, but rather rapidly approximate to exact solutions. Just as in the case of finding an approximate extremum of a function, where the choice of the initial point determines the final result (the difference is seen in high-precision numbers), the starting point of the genetic algorithm in solving the TSP determines the route returned by the algorithm (the difference is seen as in factually differing routes, as well as in their lengths) [36, 15, 39, 24].

The genetic algorithm is a multistep iterative procedure that at every step (iteration) expunges low-quality solution candidates and gathers high-quality ones [52, 5, 4, 37, 28]. As the TSP size increases (i. e., the number of nodes increases), the solution candidate (feasible route) becomes bigger (counting in nodes to be passed). Thus, the genetic algorithm iteration is run slower as the TSP size increases [38, 20, 37, 21, 18]. Therefore, the genetic algorithms for finding approximately shortest routes are limited to some number of nodes, at which the approximate solution cannot be produced within reasonable amount of time. Depending on the time resource reasonability, this number may be a few tens of thousands of nodes or so. This is the limitation of the fastest TSP solution approximation. So, even this approximation is unlikely to be applicable for solving extremely big-sized TSPs with a few hundred thousands to millions of nodes [25, 36, 46, 9, 33, 34].

2. Motivation and goal

The genetic algorithm is a type of the heuristic algorithm which is easily fine-tuned [4, 24]. For instance, it is easier even compared to fine-tuning shallow neural networks, let alone deep neural networks [33]. The fine-tuning implies optimal selection of the maximal number of iterations and properties of genetic mutations. As a result, the genetic algorithm converges a little bit faster by the rationalized mutations and stops appropriately either at the maximal number of iterations or by using this number to calculate the early stop. Although it is uncertain how close the exactly shortest route and approximately shortest route lengths are, it is highly probable that the latter are at most 3 % away from the minimum for a fine-tuned genetic algorithm [1, 38, 29, 39, 24, 48, 23]. However, even a fine-tuned genetic algorithm cannot solve (extremely) big-sized TSPs within reasonable amount of time. In particular, this can be checked by finding self-intersections of the approximate route.

The TSP approximation can be made faster by dividing the task of finding the shortest closed-loop route into two or more subtasks of finding the shortest open-loop subroutes whose union is the shortest closed-loop route. In this way, the initial TSP is substituted by a set of open-loop TSPs which are smaller subproblems (compared to the initial closed-loop TSP). Division into smaller-sized subproblems resembles the evolutionary divide and conquer approach, by which a genetic algorithm explores the space of problem subdivisions rather than the space of solutions themselves [50]. The smaller-sized TSPs are expected to be solved faster by the genetic (or some other) algorithm, and that is where the process of solving the bigger-sized TSP can be sped up. The matter is how the division is to be actually fulfilled, and how the open-loop subroutes are to be assembled into the (approximately) shortest route of the initial TSP [18, 11].

The most obvious answer is to cluster the nodes of a closed-loop TSP [33, 51, 12]. In the simplest case, the nodes are divided into two clusters, one of which contains the node whence the salesman starts one's tour. This cluster must also contain a node at which the respective open-loop subroute ends. In other words, it is the destination node of the starting subroute. This node serves as a starting node for the subsequent subproblem. In general, whichever the number of clusters is, the respective smaller-sized subproblems are open-loop TSPs, whose starting and destination nodes are different. The depot (a node of the salesman's starting departure and the ending arrival in the initial closed-loop TSP) is the starting node of the first open-loop TSP, and it is the destination node of the last open-loop TSP whose subroute must complete the initial TSP route. The subroutes must share only nodes at which they are assembled into the closed-loop route.

Having divided the nodes into a definite number of clusters, it is then possible to parallelize the process of solving the initial TSP (for simpler distinction, let it be called the whole problem in further consideration).

Unlike the parallel tabu search algorithm [9], where several moves are performed concurrently requiring some synchronization, the parallelization herein is implied in multiple set-ups [44, 27]. The parallelization set-up is determined by how many processor cores or parallel computers can be simultaneously used to run the process of solving open-loop TSPs on them. The set-up when only one open-loop TSP can be solved on a single machine is not excluded. This case does not cancel the parallelization, though. By the reason mentioned above, it is naturally expected that solving open-loop TSPs in a sequence, subproblem by subproblem, will take less computational time than solving the whole TSP. Thus, a TSP approximate solution can be obtained faster even by this, the "least parallel", parallelization set-up [44]. An additional speedup may be achieved by selecting the most appropriate number of clusters [7, 31].

Therefore, the TSP parallelization by a set-up is efficient if the amount of computational resources spent on solving the subproblems according to this set-up is less than that spent on solving the whole problem. Along with reducing the usage of computational resources, the assembled route is supposed to be not much longer than the route in the solution of the whole problem. At least, the difference must be tolerable despite being mostly uncertain. Henceforward, the goal is to suggest an efficient method of the fast approximation of the TSP shortest route by parallelizing its solving, while the difference between the exactly minimal route length and the approximated one is uncertain. The efficiency criterion is to reduce the usage of computational resources (in particular, to shorten the computational time) by lengthening the assembled route at most by some tolerance. For achieving the goal, the following eight tasks are to be fulfilled:

1. To describe TSP variables, flags, constraints, and objective.
2. To describe the open-loop TSP.
3. To suggest a method by which the initial TSP is broken into open-loop TSPs the quickest and the initial TSP solution is efficiently assembled from the solutions of the open-loop TSPs.
4. To describe basic steps in the genetic algorithm as a solver of the TSP and open-loop TSP.
5. To describe how solving the initial TSP is parallelized and the approximately shortest route is subsequently assembled.
6. To obtain statistics of the parallelization performance in order to estimate its efficiency.
7. To discuss the parallelization efficiency and limitations.
8. To conclude on the scientific and practical contribution.

The paper, whose structure is directly based on the list of these tasks, proceeds as follows. Section 3 presents a formalization of the TSP and its objective along with variables, flags, and constraints used in it. The open-loop TSP is formalized in Section 4. Section 5 presents a pattern by which the initial TSP is broken into open-loop TSPs the quickest, whereupon these open-loop TSPs are solved and an approximate solution of the initial TSP is efficiently assembled from the solutions of the open-loop TSPs. The genetic algorithm as a solver of the TSP and open-loop TSP is described in Section 6. Section 7 describes how solving the initial TSP is parallelized and the approximately shortest route is subsequently assembled. The efficiency of the suggested approach is estimated in Section 8. It is further discussed in Section 9. Section 10 concludes on the scientific and practical contribution.

3. TSP variables, flags, constraints, objective

In the flat TSP of N nodes, where the depot has number 1, the salesman's route has only two-coordinate node locations, without ascensions or descents [38, 39, 34, 2, 49], and

$$\mathrm{N} = \left\{ \begin{bmatrix} p_{k1} & p_{k2} \end{bmatrix} \right\}_{k=1}^{N} \tag{1}$$

is the set of nodes with horizontal p_{k1} and vertical p_{k2} components of the location of node k. If the salesman visits node j directly after node k, then this is flagged as $x_{kj} = 1$; otherwise $x_{kj} = 0$. Moreover, to avoid surplus flagging, the direct connection of nodes k and j is flagged only in one direction (by which the salesman factually goes):

$$x_{jk} = 0 \ \text{ if } \ x_{kj} = 1 \ \text{ and } \ x_{kj} = 0 \ \text{ if } \ x_{jk} = 1. \tag{2}$$

So, the flags showing which nodes are connected, are binary:

$$x_{kj} \in \{0,\ 1\} \quad \text{by} \quad k = \overline{1,\ N} \quad \text{and} \quad j = \overline{1,\ N} \tag{3}$$

and

$$x_{kk} = 0 \quad \forall\, k = \overline{1,\ N}. \tag{4}$$

In fact, flags (3) by (2) and (4) are variables in the TSP.

There is only one departure from node k towards only one following node that is constrained by an equality

$$\sum_{j=1}^{N} x_{kj} = 1 \quad \forall\, k = \overline{1,\ N}. \tag{5}$$

Symmetrically, there is only one arrival at node j from only one node that is constrained by an equality

$$\sum_{k=1}^{N} x_{kj} = 1 \quad \forall\, j = \overline{1,\ N}. \tag{6}$$

To exclude closed-loop subtours so that the route be a single tour and not a union of smaller tours, the third constraint is

$$\sum_{k\in Q}\sum_{j\in Q} x_{kj} \leqslant |Q| - 1 \quad \forall\, Q \subset \left\{\overline{1,\ N}\right\} \quad \text{by} \quad 2 \leqslant |Q| < N. \tag{7}$$

If the salesman visits node j directly after node k, it is accomplished by a straight line with some constant speed. Then

$$\rho\,(k,\ j) = \sqrt{(p_{k1} - p_{j1})^2 + (p_{k2} - p_{j2})^2} = \rho\,(j,\ k) \quad \text{by} \quad k = \overline{1,\ N-1} \quad \text{and} \quad j = \overline{k+1,\ N} \tag{8}$$

is the distance covered by the salesman between nodes k and j. There are $0.5N\,(N-1)$ nonzero symmetric distances (8)

$$\left\{\{\rho\,(k,\ j)\}_{k=1}^{N-1}\right\}_{j=k+1}^{N} \tag{9}$$

in the TSP of N nodes. Distances (9) can be mapped into time or other units implying the cost of completing the TSP route. The respective objective function

$$\rho_\Sigma\left(\left\{\{x_{kj}\}_{k=1}^{N}\right\}_{j=1}^{N}\right) = \sum_{k=1}^{N}\sum_{j=1}^{N} x_{kj}\cdot\rho\,(k,\ j) \tag{10}$$

is to be minimized subject to constraints (2)—(7). The minimization goal is to find such flags

$$x_{kj}^{*} \in \{0,\ 1\} \quad \text{for} \quad k = \overline{1,\ N} \quad \text{and} \quad j = \overline{1,\ N} \tag{11}$$

at which

$$\sum_{k=1}^{N}\sum_{j=1}^{N} x_{kj}^{*}\cdot\rho\,(k,\ j) = \rho_\Sigma\left(\left\{\{x_{kj}^{*}\}_{k=1}^{N}\right\}_{j=1}^{N}\right) = \rho_\Sigma^{*} =$$

$$= \min_{\left\{\{x_{kj}\}_{k=1}^{N}\right\}_{j=1}^{N}} \sum_{k=1}^{N}\sum_{j=1}^{N} x_{kj}\cdot\rho\,(k,\ j). \tag{12}$$

Flags (11) give the shortest route length ρ_Σ^{*} by (12). In addition, flags (11) allow building a minimum-length route (an optimal route). Nevertheless, minimum (12) can be reached by more than one set

$$\left\{\{x_{kj}^{*}\}_{k=1}^{N}\right\}_{j=1}^{N} \tag{13}$$

of flags (11). So, the TSP may have multiple minimum-length routes having the same length ρ_Σ^{*} [25, 1, 34, 2, 49].

4. Open-loop TSP

In the open-loop TSP of N nodes (1), the salesman must depart from the depot (node 1) and arrive at node N (the destination node). There is only one departure from node k being not node N towards only one following node that is constrained by an equality

$$\sum_{j=1}^{N} x_{kj} = 1 \ \ \forall k = \overline{1,\ N-1}. \tag{14}$$

Symmetrically, there is only one arrival at node j from only one node being not node N that is constrained by an equality

$$\sum_{k=1}^{N-1} x_{kj} = 1 \ \ \forall j = \overline{1,\ N}. \tag{15}$$

Closed-loop subtours in the open-loop TSP are excluded by constraint (7) as well. The respective objective function (10) is to be minimized subject to constraints (2) — (4), (7), (14), (15).

5. Rectangular cell clustering pattern

Nodes (1) of an initial TSP can be broken into multiple groups (clusters) by applying a clustering method. In particular, it can be a method of *K*-means or *K*-medoids [40, 14]. Then each of the clusters correspond to an open-loop TSP. Breaking into just two clusters is the quickest. As the number of clusters is increased, the clustering is performed slower. The slowdown becomes more significant as the number of nodes increases.

As the first two clusters are obtained by the quickest clustering, they can be further clustered. Each of the clusters is broken into two clusters, which is fulfilled the quickest. Such a clustering process can be kept until it is enough — the open-loop TSPs become sufficiently small. There are $M = 2^n$ clusters by $n \in \mathbb{N}\setminus\{1\}$. At step n of the clustering, these clusters are

$$\mathrm{N}_m(n) = \left\{\left[\ p_{k_m 1} \quad p_{k_m 2}\ \right]\right\}_{k_m \in K_m(n)} \ \text{ for } \ m = \overline{1,\ M} \tag{16}$$

by

$$K_1(n) \subset K_1(n-1), \ \ 1 \in K_1(n), \tag{17}$$

$$\begin{gathered} K_{2i-1}(n) \cup K_{2i}(n) = K_i(n-1) \ \text{ and } \ K_{2i-1}(n) \cap K_{2i}(n) = \emptyset, \\ \mathrm{N}_{2i-1}(n) \cup \mathrm{N}_{2i}(n) = \mathrm{N}_i(n-1) \ \text{ and } \ \mathrm{N}_{2i-1}(n) \cap \mathrm{N}_{2i}(n) = \emptyset \ \text{ for } \ i = \overline{1,\ 2^{n-1}}, \end{gathered} \tag{18}$$

and

$$\mathrm{N}_1(1) = \left\{\left[\ p_{k_1 1} \quad p_{k_1 2}\ \right]\right\}_{k_1 \in K_1(1)}, \ \ K_1(1) \subset \left\{\overline{1,\ N}\right\}, \ \ 1 \in K_1(1), \tag{19}$$

$$\mathrm{N}_2(1) = \left\{\left[\ p_{k_2 1} \quad p_{k_2 2}\ \right]\right\}_{k_2 \in K_2(1)}, \ \ K_2(1) \subset \left\{\overline{2,\ N}\right\}, \tag{20}$$

where

$$\begin{gathered} K_1(1) \cup K_2(1) = \left\{\overline{1,\ N}\right\} \ \text{ and } \ K_1(1) \cap K_2(1) = \emptyset, \\ \mathrm{N}_1(1) \cup \mathrm{N}_2(1) = \mathrm{N} \ \text{ and } \ \mathrm{N}_1(1) \cap \mathrm{N}_2(1) = \emptyset. \end{gathered} \tag{21}$$

Obviously, these clusters are non-overlapping, so

$$\begin{gathered} \bigcup_{m=1}^{M} K_m = \left\{\overline{1,\ N}\right\} \ \text{ and } \ \bigcup_{m=1}^{M} \mathrm{N}_m = \mathrm{N}, \\ K_{m_1} \cap K_{m_2} = \emptyset \ \text{ for } \ m_1 = \overline{1,\ M} \ \text{ and } \ m_2 = \overline{1,\ M} \ \text{ by } \ m_1 \neq m_2. \end{gathered} \tag{22}$$

It is clear that the clusters are broken quicker as number n is increased.

An example of 250000 nodes clustered in 16 groups is shown in Figure 1, where the depot is marked as square, and the centroids of the clusters are marked as circles. The 16 clusters appear roughly squarish. The same set divided into 64 clusters is shown in Figure 2. The 64 clusters now appear far less squarish, even roughly. Nevertheless, the clusters in Figure 1 fit the square (rectangular) lattice pattern shown in Figure 3.

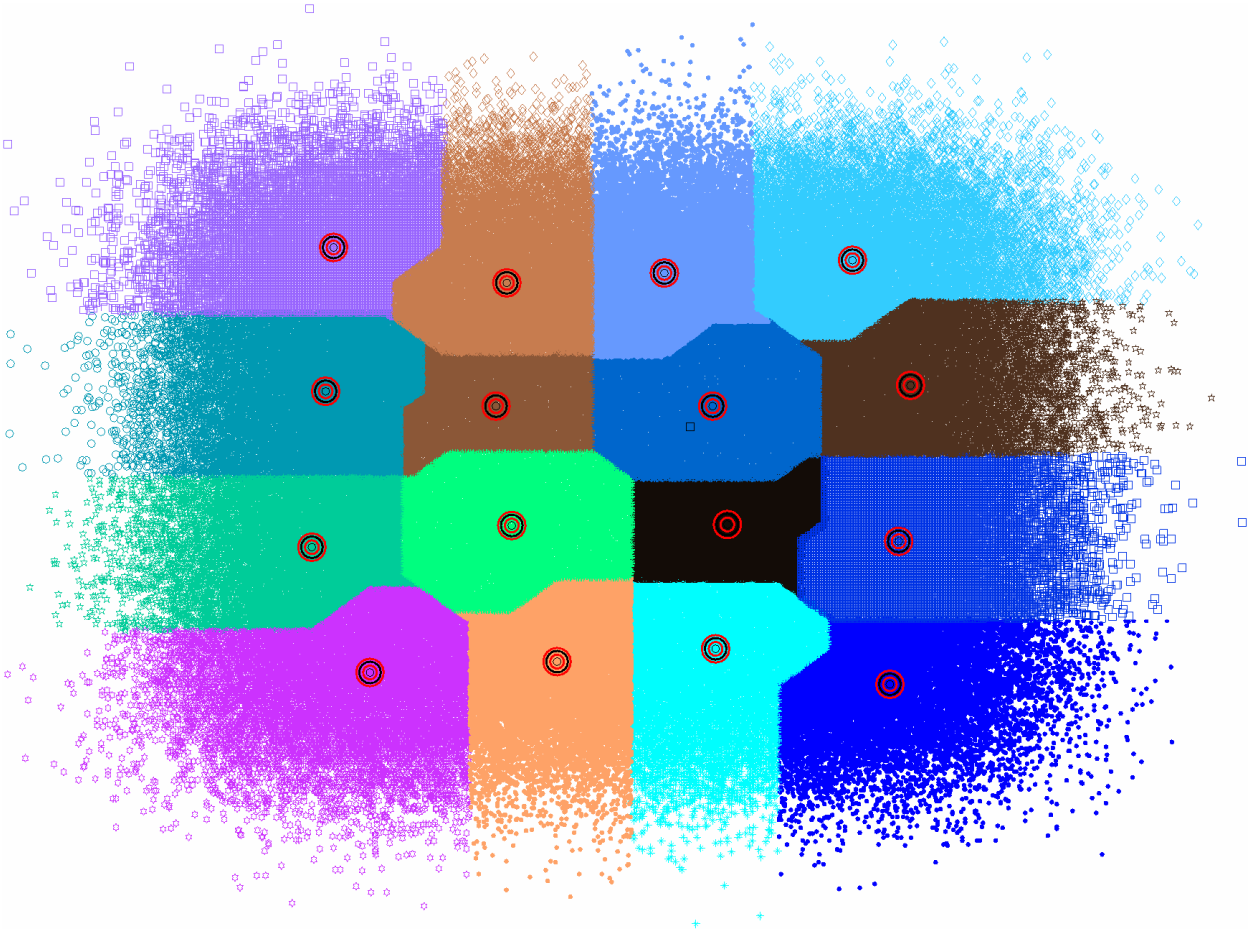

Figure 1. A set of 250000 nodes divided into 16 clusters

In general, centroids

$$\left\{\mathbf{C}_m = \left[\begin{array}{cc} c_{m1} & c_{m2} \end{array}\right]\right\}_{m=1}^{M} \tag{23}$$

of real clusters are only approximately close to the cell centers but generally do not coincide with them. As the number of clusters is too increased, the respective rectangular lattice pattern becomes inappropriate (Figure 2). So, there is a number of clusters (or number n), for which the pattern and the factual clustering result are approximately the closest. This implies that, instead of the iterative clustering by (16) — (22), the 2^n clusters can be made by just the rectangular cell clustering pattern (exemplified for 16 clusters in Figure 3). The initial set of nodes (1) belongs to rectangle

$$\left[\min_{k=\overline{1,\,N}} p_{k1};\ \max_{k=\overline{1,\,N}} p_{k1}\right] \times \left[\min_{k=\overline{1,\,N}} p_{k2};\ \max_{k=\overline{1,\,N}} p_{k2}\right]. \tag{24}$$

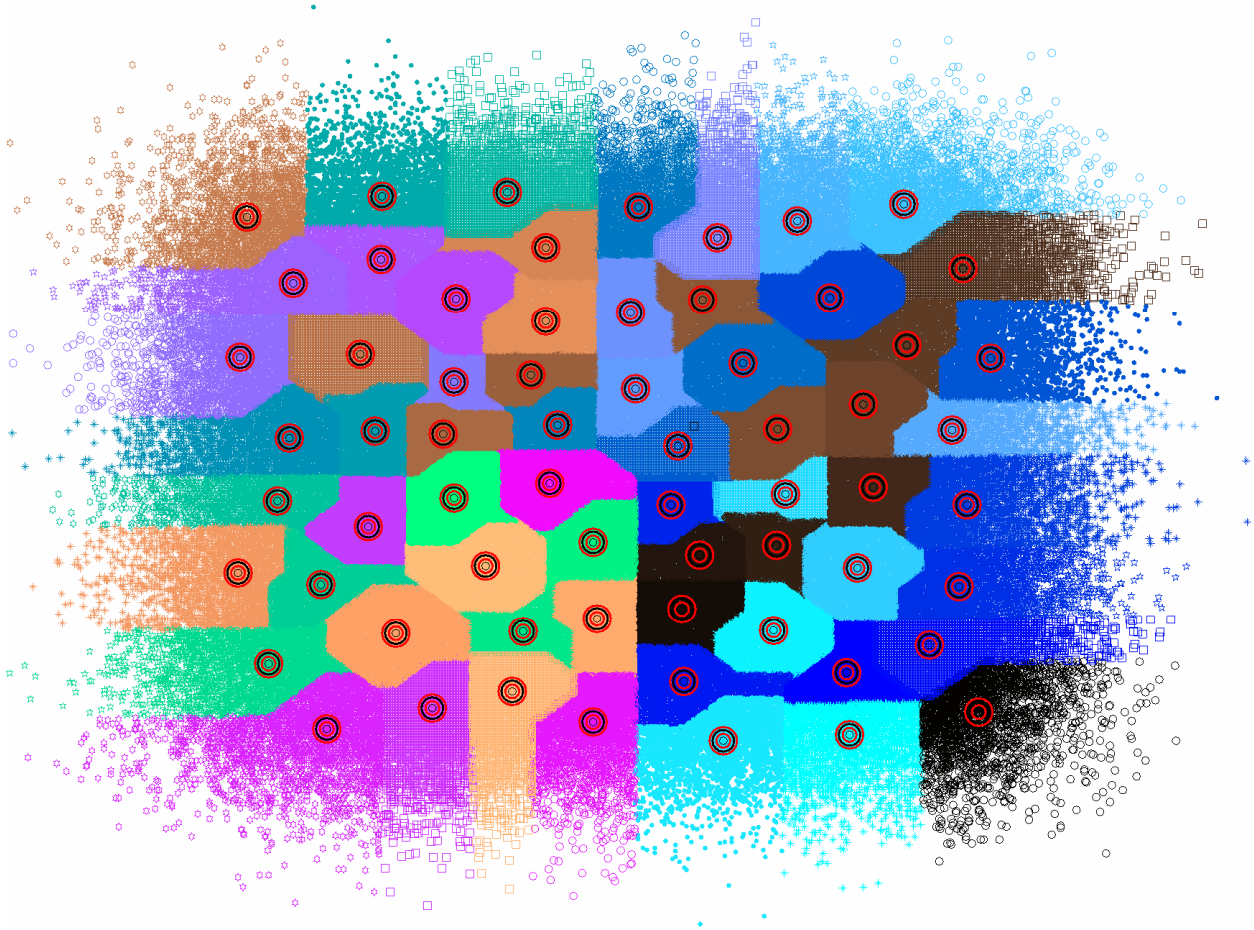

Figure 2. The set of 250000 nodes from Figure 1 divided into 64 clusters

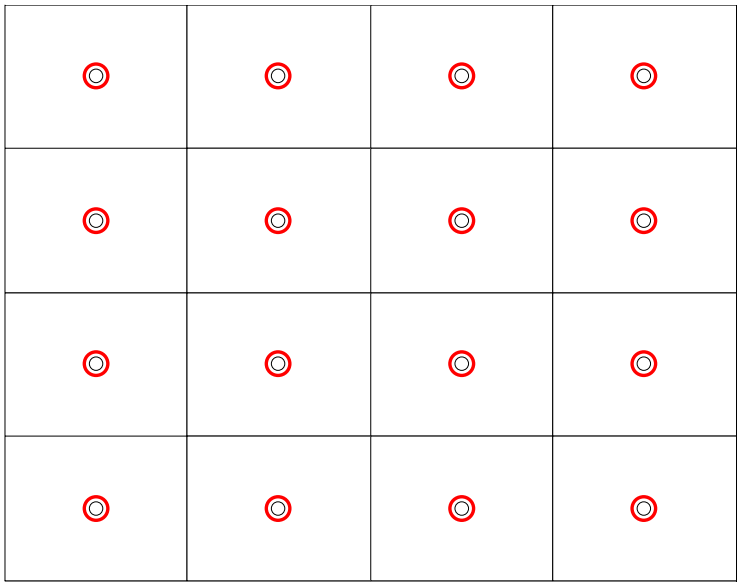

Figure 3. An example of the square (rectangular) lattice pattern of 16 clusters

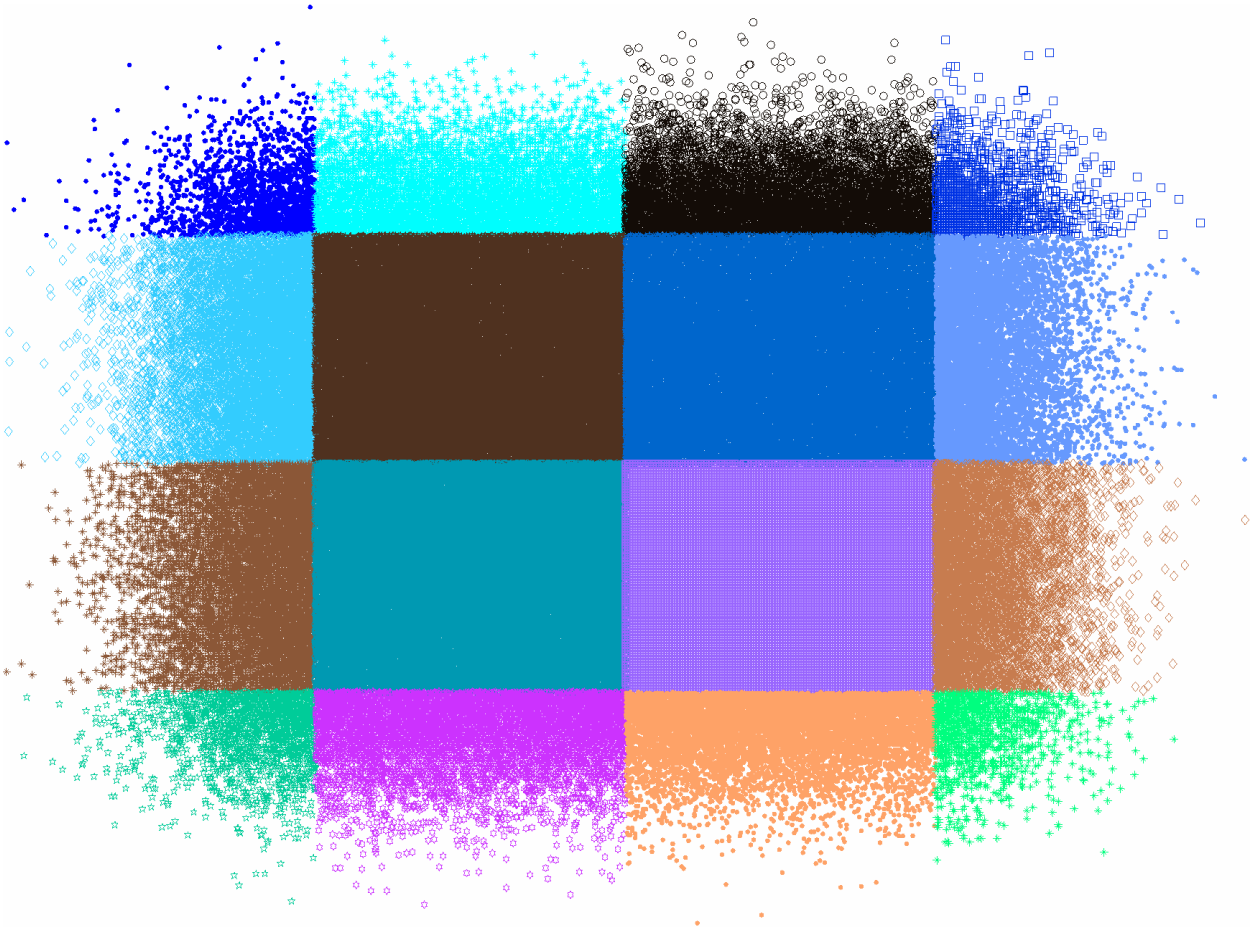

Figure 4. The set of 250000 nodes from Figure 1 (the depot is not marked) divided into 16 clusters by the rectangular cell clustering pattern

If the lattice is of M_{hor} horizontal and M_{vert} vertical cells, where $M = M_{\text{hor}} \cdot M_{\text{vert}}$, rectangle (24) is uniformly broken into $M = 2^n$ cells (subrectangles)

$$[a_{m_{\text{hor}}m_{\text{vert}}};\ b_{m_{\text{hor}}m_{\text{vert}}}] \times [g_{m_{\text{hor}}m_{\text{vert}}};\ h_{m_{\text{hor}}m_{\text{vert}}}] =$$

$$= \left[\left(\min_{k=\overline{1,\ N}} p_{k1} \right) + (m_{\text{hor}} - 1) \cdot \frac{\max\limits_{k=\overline{1,\ N}} p_{k1} - \min\limits_{k=\overline{1,\ N}} p_{k1}}{M_{\text{hor}}};\ \left(\min_{k=\overline{1,\ N}} p_{k1} \right) + m_{\text{hor}} \cdot \frac{\max\limits_{k=\overline{1,\ N}} p_{k1} - \min\limits_{k=\overline{1,\ N}} p_{k1}}{M_{\text{hor}}} \right] \times$$

$$\times \left[\left(\min_{k=\overline{1,\ N}} p_{k2} \right) + (m_{\text{vert}} - 1) \cdot \frac{\max\limits_{k=\overline{1,\ N}} p_{k2} - \min\limits_{k=\overline{1,\ N}} p_{k2}}{M_{\text{vert}}};\ \left(\min_{k=\overline{1,\ N}} p_{k2} \right) + m_{\text{vert}} \cdot \frac{\max\limits_{k=\overline{1,\ N}} p_{k2} - \min\limits_{k=\overline{1,\ N}} p_{k2}}{M_{\text{vert}}} \right] \quad (25)$$

for $m_{\text{hor}} = \overline{1,\ M_{\text{hor}}}$ and $m_{\text{vert}} = \overline{1,\ M_{\text{vert}}}$. The clusters in this case are created by just checking whether a node belongs to the respective cell (25). An example of applying the rectangular cell clustering pattern to the set of nodes in Figure 1 is shown in Figure 4. Despite the rectangular clusters in Figure 4 do differ from those in Figure 1 (although the "boundary" clusters in these Figures are nearly similar), they are obtained a few hundred times faster. Thus, the division in Figure 4 takes up to 0.29 seconds, whereas the division in Figure 1 takes about 20 seconds. Dividing a million nodes into such a square of 16 clusters by the rectangular cell clustering pattern takes up to 1.3 seconds, and applying the iterative clustering by (16) — (22) takes more than 2 minutes in this case.

Every cluster corresponds to its respective open-loop TSP. Its (approximately) shortest open-loop route should be assembled into the (approximately) shortest route of the initial TSP. Owing to the rectangular lattice consisting of 2^n cells, the assembling is quite easy being a symmetric rectangular closed-loop serpentine (Figure 5).

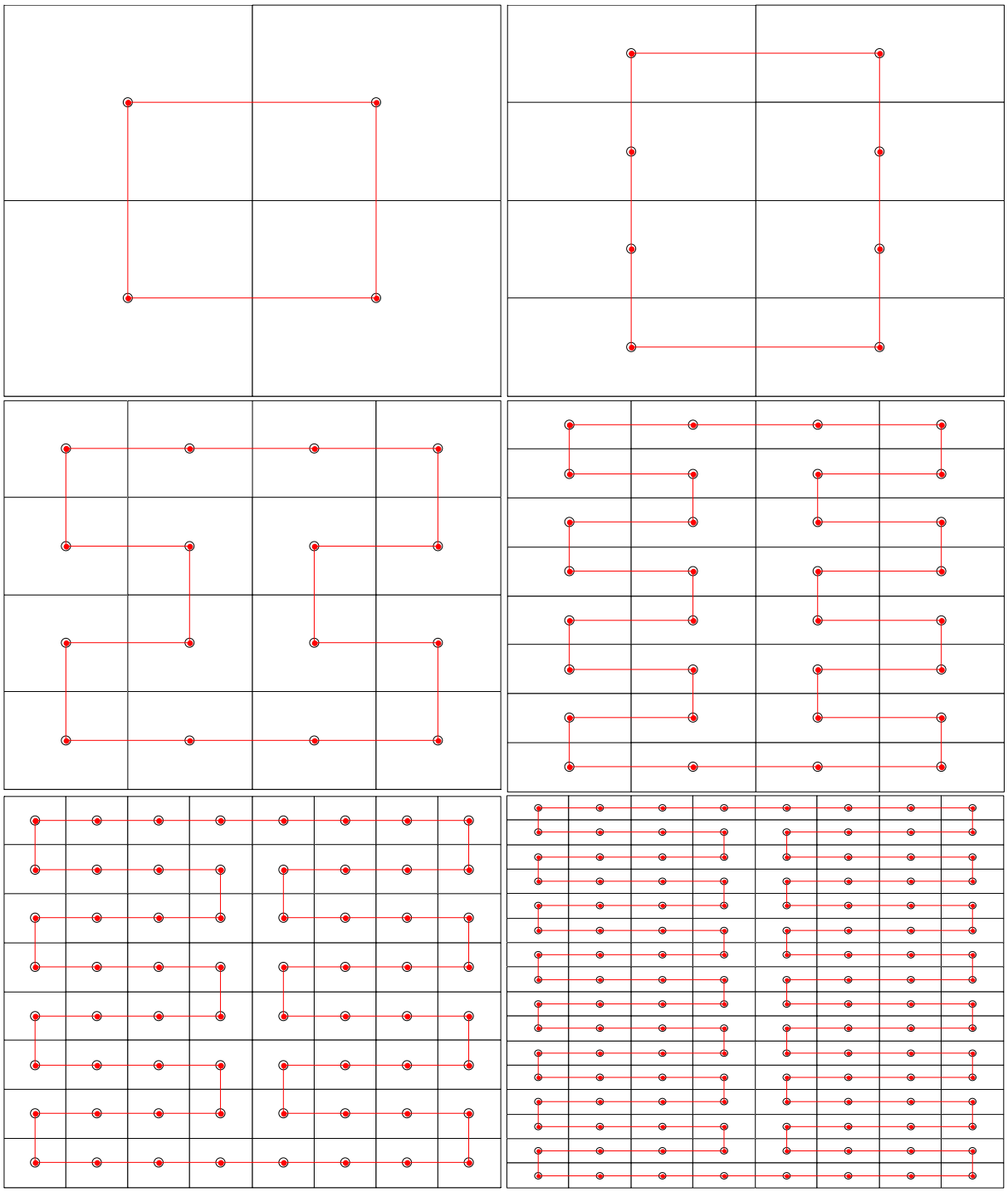

Figure 5. Assembling the open-loop routes by the serpentine patterns of the shortest route passing through centers of the lattice cells (4, 8, 16, 32, 64, 128 cells)

It is worth noting that the serpentine patterns in Figure 5, except for the 4 and 8 cells, are not the only possible ones. There are other, symmetric and non-symmetric, patterns whose lengths are the same equal to the length of the respective pattern in Figure 5. Obviously, when the cell is a unit square, the length is 2^n units.

6. Genetic algorithm

The solver of both the TSP and open-loop TSP is the genetic algorithm principally requiring at its input a set of node locations (1), the depot location $\begin{bmatrix} p_{11} & p_{12} \end{bmatrix}$, a population size, and a set of mutation operators. The other auxiliary options are usually set at their default values. The population is a series of pseudorandom routes called chromosomes. For the genetic algorithm of solving an ordinary (closed-loop) TSP, each element of the population is an $(N-1)$-dimensional vector

$$\mathbf{S} = [n_h]_{1\times(N-1)} \quad \text{by} \quad n_h \in \{\overline{2,\ N}\} \quad \forall\, h = \overline{1,\ N-1} \tag{26}$$

of non-depot nodes the salesman should visit. For every route of the population, the following routine is executed during an iteration of the algorithm. First, the distance to the node following the depot is calculated as

$$d = \rho\,(1,\ n_1) \tag{27}$$

by (8). Second, the remaining distances except the last one are accumulated into the running variable d:

$$d_{\text{obs}} = d, \quad d = d_{\text{obs}} + \rho\,(n_h,\ n_{h+1}) \quad \text{for} \quad h = \overline{1,\ N-2}. \tag{28}$$

Third, the distance of returning to the depot is included the last:

$$d_{\text{obs}} = d, \quad d = d_{\text{obs}} + \rho\,(n_{N-1},\ 1)\,. \tag{29}$$

Then, the accumulated distance covered by route (26) is calculated as

$$\tilde{\rho}_\Sigma\,(\mathbf{S}) = d \geqslant \rho^*_\Sigma \tag{30}$$

and minimized over the population. Inequality (30) is the relationship between the length of a heuristically found route (26) and the exactly shortest route length from an exact solution to problem (12).

A new population is generated using mutation operators of slide, flip, swap, and crossover within a subpopulation of the currently best chromosomes. First, the slide operator moves the last node from each chromosome to the beginning of another one. Next, the flip operator swaps a random sequence of nodes inside a chromosome: a sequence

$$\left\{n_h^{(r)}\right\}_{h=h_1+1}^{h_2} \subset \left\{n_h^{(r)}\right\}_{h=1}^{N-1} \tag{31}$$

for random indices h_1 and h_2 by

$$0 \leqslant h_1 < h_2 \leqslant N-1 \tag{32}$$

from a route

$$\mathbf{S}^{(r)} = \left[n_h^{(r)}\right]_{1\times(N-1)} \tag{33}$$

is extracted and flipped as

$$\begin{gathered}
\mathbf{S}^{(r)}_{\text{obs}} = \left[n_h^{(r)*}\right]_{1\times(N-1)} = \mathbf{S}^{(r)}, \\
\mathbf{S}^{(r)} = \left[n_h^{(r)}\right]_{1\times(N-1)} \quad \text{by} \quad n_h^{(r)} = n_h^{(r)*} \quad \forall\, h = \overline{1,\ h_1} \quad \text{and} \\
n_h^{(r)} = n_{h_2+h_1-h+1}^{(r)*} \quad \forall\, h = \overline{h_1+1,\ h_2} \quad \text{and} \quad n_h^{(r)} = n_h^{(r)*} \quad \forall\, h = \overline{h_2+1,\ N-1}.
\end{gathered} \tag{34}$$

The flip operator returns then an updated vector (33) after (34).

The swap operator selects the same-index-and-length sequence of nodes from two chromosomes (33) and

$$\mathbf{S}^{(q)} = \left[n_h^{(q)}\right]_{1\times(N-1)} \tag{35}$$

for random integers h_3 and h_4 by

$$0 \leqslant h_3 < h_4 \leqslant N-1 \tag{36}$$

and for $r \neq q$, whereupon they are interchanged. Thus, sequences

$$\left\{n_h^{(r)}\right\}_{h=h_3+1}^{h_4} \subset \left\{n_h^{(r)}\right\}_{h=1}^{N-1} \tag{37}$$

and

$$\left\{n_h^{(q)}\right\}_{h=h_3+1}^{h_4} \subset \left\{n_h^{(q)}\right\}_{h=1}^{N-1} \tag{38}$$

are interchanged as

$$\begin{gathered}\mathbf{S}_{\text{obs}}^{(r)} = \left[n_h^{(r)*}\right]_{1\times(N-1)} = \mathbf{S}^{(r)},\\ \mathbf{S}^{(r)} = \left[n_h^{(r)}\right]_{1\times(N-1)} \text{ by } n_h^{(r)} = n_h^{(r)*} \ \forall h = \overline{1,\ h_3} \text{ and}\\ n_h^{(r)} = n_h^{(q)} \ \forall h = \overline{h_3+1,\ h_4} \text{ and } n_h^{(r)} = n_h^{(r)*} \ \forall h = \overline{h_4+1,\ N-1}\end{gathered} \tag{39}$$

and

$$\begin{gathered}\mathbf{S}_{\text{obs}}^{(q)} = \left[n_h^{(q)*}\right]_{1\times(N-1)} = \mathbf{S}^{(q)},\\ \mathbf{S}^{(q)} = \left[n_h^{(q)}\right]_{1\times(N-1)} \text{ by } n_h^{(q)} = n_h^{(q)*} \ \forall h = \overline{1,\ h_3} \text{ and}\\ n_h^{(q)} = n_h^{(r)*} \ \forall h = \overline{h_3+1,\ h_4} \text{ and } n_h^{(q)} = n_h^{(q)*} \ \forall h = \overline{h_4+1,\ N-1}.\end{gathered} \tag{40}$$

The swap operator returns then updated vectors (33) and (35) after (39) and (40), respectively.

The crossover operator, somewhat resembling the swapping, takes two chromosomes (33) and (35) for $r \neq q$, and cuts each chromosome in two random parts using random integers h_r and h_q by

$$1 \leqslant h_r \leqslant N-2 \tag{41}$$

and

$$1 \leqslant h_q \leqslant N-2, \tag{42}$$

where h_r first nodes in chromosome (33) are left and h_q first nodes in chromosome (35) are left. Thereupon the chromosome parts of H nodes long by

$$1 \leqslant H \leqslant N-1-\max\{h_r,\ h_q\} \tag{43}$$

are interchanged:

$$\mathbf{S}^{(r)**} = \left[n_h^{(r)**}\right]_{1\times(N-1)} = \left\{\left\{n_h^{(r)}\right\}_{h=1}^{h_r}, \left\{n_h^{(q)}\right\}_{h=h_q+1}^{h_q+H}, \left\{n_h^{(r)}\right\}_{h=h_r+H+1}^{N-1}\right\} \tag{44}$$

and

$$\mathbf{S}^{(q)**} = \left[n_h^{(q)**}\right]_{1\times(N-1)} = \left\{\left\{n_h^{(q)}\right\}_{h=1}^{h_q}, \left\{n_h^{(r)}\right\}_{h=h_r+1}^{h_r+H}, \left\{n_h^{(q)}\right\}_{h=h_q+H+1}^{N-1}\right\}. \tag{45}$$

The crossover operator returns new (mutated) routes (44) and (45).

The algorithm for solving the open-loop TSP is slightly modified: instead of (26), each element of the population is an $(N-2)$-dimensional vector

$$\mathbf{S} = [n_h]_{1\times(N-2)} \ \text{ by } \ n_h \in \{\overline{2,\ N-1}\} \ \ \forall\, h = \overline{1,\ N-2} \tag{46}$$

of non-start-end nodes the salesman should visit (where N is the number of nodes in an open-loop TSP, and the salesman should start its route at node 1 and complete at node N). The distance to the node following the depot is calculated as (27) and the remaining distances except the last one are accumulated into the running variable d:

$$d_{\text{obs}} = d, \ \ d = d_{\text{obs}} + \rho\,(n_h,\ n_{h+1}) \ \ \text{for} \ \ h = \overline{1,\ N-3}. \tag{47}$$

The distance to node N is included the last:

$$d_{\text{obs}} = d, \ \ d = d_{\text{obs}} + \rho\,(n_{N-2},\ N)\,. \tag{48}$$

Then, the accumulated distance covered by route (46) is calculated as (30). Instead of (31) — (45), a new population is similarly generated by

$$\left\{n_h^{(r)}\right\}_{h=h_1+1}^{h_2} \subset \left\{n_h^{(r)}\right\}_{h=1}^{N-2}, \tag{49}$$

$$\mathbf{S}^{(r)} = \left[n_h^{(r)}\right]_{1\times(N-2)}, \tag{50}$$

$$0 \leqslant h_1 < h_2 \leqslant N-2, \tag{51}$$

$$\begin{gathered}
\mathbf{S}_{\text{obs}}^{(r)} = \left[n_h^{(r)*}\right]_{1\times(N-2)} = \mathbf{S}^{(r)}, \\
\mathbf{S}^{(r)} = \left[n_h^{(r)}\right]_{1\times(N-2)} \ \text{ by } \ n_h^{(r)} = n_h^{(r)*} \ \ \forall\, h = \overline{1,\ h_1} \ \text{ and } \\
n_h^{(r)} = n_{h_2+h_1-h+1}^{(r)*} \ \ \forall\, h = \overline{h_1+1,\ h_2} \ \text{ and } \ n_h^{(r)} = n_h^{(r)*} \ \ \forall\, h = \overline{h_2+1,\ N-2},
\end{gathered} \tag{52}$$

$$0 \leqslant h_3 < h_4 \leqslant N-2, \tag{53}$$

$$\mathbf{S}^{(q)} = \left[n_h^{(q)}\right]_{1\times(N-2)}, \tag{54}$$

$$\left\{n_h^{(r)}\right\}_{h=h_3+1}^{h_4} \subset \left\{n_h^{(r)}\right\}_{h=1}^{N-2}, \tag{55}$$

$$\left\{n_h^{(q)}\right\}_{h=h_3+1}^{h_4} \subset \left\{n_h^{(q)}\right\}_{h=1}^{N-2}, \tag{56}$$

$$\begin{gathered}
\mathbf{S}_{\text{obs}}^{(r)} = \left[n_h^{(r)*}\right]_{1\times(N-2)} = \mathbf{S}^{(r)}, \\
\mathbf{S}^{(r)} = \left[n_h^{(r)}\right]_{1\times(N-2)} \ \text{ by } \ n_h^{(r)} = n_h^{(r)*} \ \ \forall\, h = \overline{1,\ h_3} \ \text{ and } \\
n_h^{(r)} = n_h^{(q)} \ \ \forall\, h = \overline{h_3+1,\ h_4} \ \text{ and } \ n_h^{(r)} = n_h^{(r)*} \ \ \forall\, h = \overline{h_4+1,\ N-2},
\end{gathered} \tag{57}$$

$$\begin{gathered}
\mathbf{S}_{\text{obs}}^{(q)} = \left[n_h^{(q)*}\right]_{1\times(N-2)} = \mathbf{S}^{(q)}, \\
\mathbf{S}^{(q)} = \left[n_h^{(q)}\right]_{1\times(N-2)} \ \text{ by } \ n_h^{(q)} = n_h^{(q)*} \ \ \forall\, h = \overline{1,\ h_3} \ \text{ and } \\
n_h^{(q)} = n_h^{(r)*} \ \ \forall\, h = \overline{h_3+1,\ h_4} \ \text{ and } \ n_h^{(q)} = n_h^{(q)*} \ \ \forall\, h = \overline{h_4+1,\ N-2},
\end{gathered} \tag{58}$$

$$\mathbf{S}^{(r)**}=\left[n_h^{(r)**}\right]_{1\times(N-2)}=\left\{\left\{n_h^{(r)}\right\}_{h=1}^{h_r},\ \left\{n_h^{(q)}\right\}_{h=h_q+1}^{h_q+H},\ \left\{n_h^{(r)}\right\}_{h=h_r+H+1}^{N-2}\right\} \tag{59}$$

and

$$\mathbf{S}^{(q)**}=\left[n_h^{(q)**}\right]_{1\times(N-2)}=\left\{\left\{n_h^{(q)}\right\}_{h=1}^{h_q},\ \left\{n_h^{(r)}\right\}_{h=h_r+1}^{h_r+H},\ \left\{n_h^{(q)}\right\}_{h=h_q+H+1}^{N-2}\right\} \tag{60}$$

by

$$1\leqslant h_r\leqslant N-3,\ \ 1\leqslant h_q\leqslant N-3,\ \ 1\leqslant H\leqslant N-2-\max\left\{h_r,\ h_q\right\}, \tag{61}$$

respectively.

7. Parallelization and assembling the approximately shortest route

The shortest route passing through the cells of the rectangular lattice pattern is easily assembled by using the following routine. The cell centers can be numbered starting from the left top corner downwards (see, e. g., Figure 6 and Figure 7, where the lattices with 8 and 32 cells are shown in two versions). If $\mathbf{U}=[u_m]_{1\times M}$ is the shortest route, then

$$u_m=m\ \text{ for }\ m=\overline{1,\ M_{\text{hor}}}, \tag{62}$$

$$u_{mM_{\text{hor}}+q}=2mM_{\text{hor}}-q+1\ \text{ for }\ q=\overline{1,\ 0.5M_{\text{hor}}}\ \text{ and }\ m=\overline{1,\ 0.5M_{\text{vert}}-1}, \tag{63}$$

$$u_{mM_{\text{hor}}+0.5M_{\text{hor}}+q}=(2m+1)\,M_{\text{hor}}-0.5M_{\text{hor}}+q\ \text{ for }\ q=\overline{1,\ 0.5M_{\text{hor}}}\ \text{ and }\ m=\overline{1,\ 0.5M_{\text{vert}}-1}, \tag{64}$$

$$u_{0.5M+m}=M-m+1\ \text{ for }\ m=\overline{1,\ M_{\text{hor}}}, \tag{65}$$

$$u_{0.5M+mM_{\text{hor}}+q}=(M_{\text{vert}}-2m+1)\,M_{\text{hor}}-M_{\text{hor}}+q\ \text{ for }\ q=\overline{1,\ 0.5M_{\text{hor}}}\ \text{ and }\ m=\overline{1,\ 0.5M_{\text{vert}}-1}, \tag{66}$$

$$\begin{gathered}u_{0.5M+mM_{\text{hor}}+0.5M_{\text{hor}}+q}=(M_{\text{vert}}-2m)\,M_{\text{hor}}-0.5M_{\text{hor}}-q+1\\ \text{for }\ q=\overline{1,\ 0.5M_{\text{hor}}}\ \text{ and }\ m=\overline{1,\ 0.5M_{\text{vert}}-1}.\end{gathered} \tag{67}$$

It is clear that if $\sqrt{M}$ is integer then $M_{\text{hor}}=M_{\text{vert}}$.

Henceforward, the approximately shortest route by the rectangular lattice pattern is assembled in accordance with (62)—(67). The assembling is started with the cluster containing the depot. Prior to the assembling, the open-loop TSPs (subproblems) can be solved in any succession (e. g., sequentially, subproblem by subproblem) or in parallel (simultaneously, on parallel processor cores or on parallel computers). Thus, the sequential parallelization is distinguished from the in-parallel parallelization implying the way of coherence and simultaneousness of solving the open-loop TSPs. Meanwhile, there is no requirement of any sort of synchronization.

The two subtours for open-loop TSP u_m and open-loop TSP u_{m+1}, $m=\overline{1,\ M-1}$, should be connected by a node from cluster u_m which is the closest to cluster u_{m+1}. For example, if the depot is in cluster 14 (see Figure 6, second row), then the starting subtour for TSP 14 should be connected by a node from cluster 14 closest to cluster 13; cluster 13 should be connected by its node closest to cluster 9; cluster 9 should be connected by its node closest to cluster 10; ...; cluster 15 should be connected by its node closest to cluster 14. Instead of searching through all the nodes of both clusters, the closest node can be approximately determined as one of the four nodes within the rectangular cell (cluster) which are the westernmost, easternmost, southernmost, northernmost. The westernmost node has the least value of its first (horizontal) component; the easternmost node has the largest value of its first (horizontal) component; the southernmost node has the least value of its second (vertical) component; the northernmost node has the largest value of its second (vertical) component.

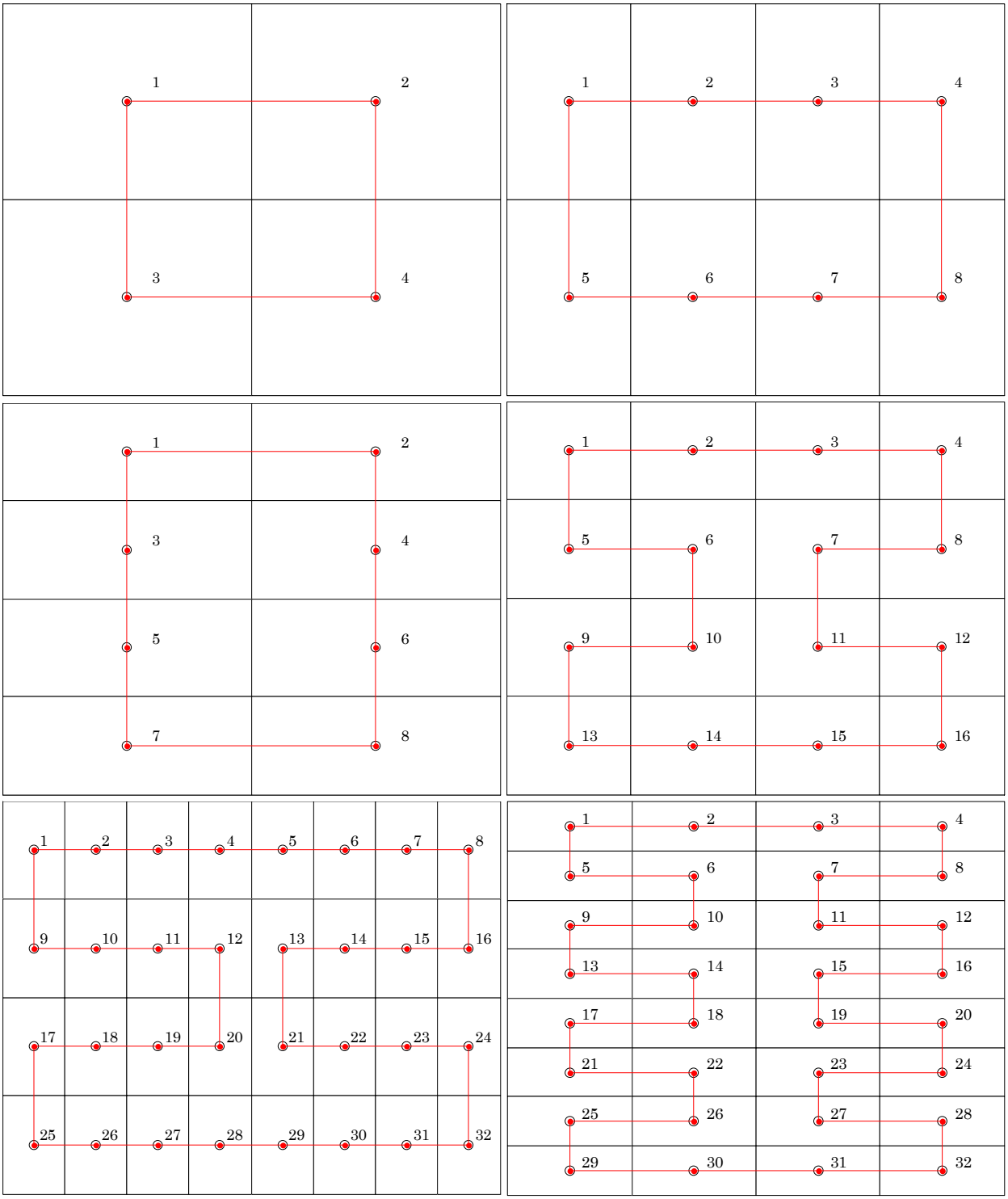

Figure 6. The rectangular lattice patterns of 4, 8 (two versions), 16, 32 (two versions) cells (see Figure 5) and the shortest route passing through the numbered centers of the lattice cells in accordance with (62) — (67)

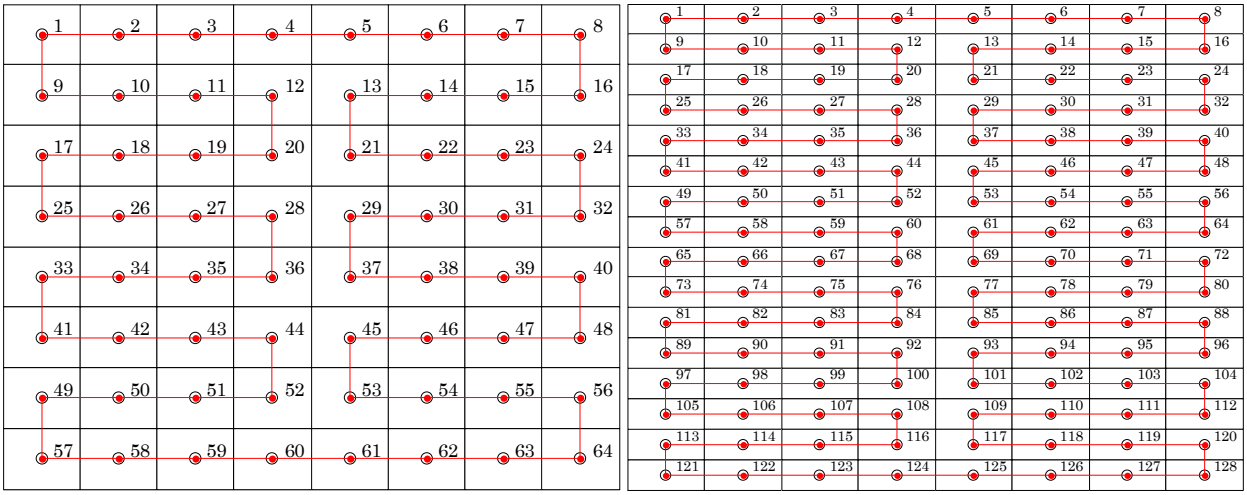

Figure 7. The rectangular lattice patterns of 64 and 128 cells (see Figure 5) and the shortest route passing through the numbered centers of the lattice cells in accordance with (62) — (67)

8. Efficiency

To see how efficient the rectangular cell clustering pattern parallelization is, the number of nodes is varied from 2001 to 8001 for

$$N \in \{2001,\ 4001,\ 6001,\ 8001\} \tag{68}$$

and three versions of the rectangular cell clustering pattern are used for

$$M \in \{4,\ 16,\ 64\}\,, \tag{69}$$

i. e. $n \in \{2,\ 4,\ 6\}$. To randomly generate nodes, pseudorandom numbers independently drawn from the standard uniform distribution on the open interval $(0;\ 1)$ and independently drawn from the standard normal distribution are used. Denote uniformly distributed variates by θ_1, θ_2 and normally distributed ones by η_1, η_2, η_3, η_4. Then the node locations are generated as

$$p_{k1} = 150 \cdot \theta_1 + 10 \cdot \eta_1 + 50 \ \text{ and } \ p_{k2} = 150 \cdot \theta_2 + 10 \cdot \eta_2 + 50 \ \text{ by } \ k = \overline{2,\ N}, \tag{70}$$

whereas the depot location is generated as

$$p_{11} = \frac{1}{N-1} \cdot \sum_{k=2}^{N} p_{k1} + 10 \cdot \eta_3 \ \text{ and } \ p_{12} = \frac{1}{N-1} \cdot \sum_{k=2}^{N} p_{k2} + 10 \cdot \eta_4. \tag{71}$$

Denote the number of nodes in cluster m by N_m, where $N_1 = |K_1\,(n)| - 1$ (the depot is not counted). So,

$$N = 1 + \sum_{m=1}^{M} N_m.$$

The maximal number of iterations for solving the whole TSP is $200 \cdot (N-1)$, and the maximal number of iterations for solving the open-loop TSP for cluster m is set similarly — it is $200 \cdot N_m$. The algorithm early stop condition is used, by which a run of the algorithm is stopped if the shortest route length does not change for a one tenth of the maximal number of iterations. To obtain reliable and stable statistical data, the whole TSP is re-generated 30 times for every (68) and (69).

Denote by $\tilde{\rho}_{\Sigma}^{*}\left(N;\ w\right)$ the shortest route length found for the w-th whole TSP generated for N nodes by (68). For the case of the rectangular cell clustering, denote the shortest length of the subroute in the open-loop TSP for cluster m by $\tilde{\rho}_{\Sigma}^{\square\exists *(m)}\left(N_m,\ M;\ w\right)$ for the w-th whole problem generated for N nodes by (68) and M clusters by (69), $w = \overline{1,\ 30}$. Then a ratio

$$g\left(N,\ M;\ w\right) = \frac{\tilde{\rho}_{\Sigma}^{*}\left(N;\ w\right)}{\sum\limits_{m=1}^{M}\tilde{\rho}_{\Sigma}^{\square\exists *(m)}\left(N_m,\ M;\ w\right)} \tag{72}$$

reflects an accuracy gain of the parallelization by using the rectangular cell clustering pattern. In addition, denote by $l_{\Sigma}^{*}\left(N;\ w\right)$ the number of iterations taken to solve the whole TSP, and denote by $l_{\Sigma}^{\square\exists *(m)}\left(N_m,\ M;\ w\right)$ the number of iterations taken to solve the open-loop TSP for cluster m. Then a ratio

$$\lambda\left(N,\ M;\ w\right) = \frac{l_{\Sigma}^{*}\left(N;\ w\right)}{\sum\limits_{m=1}^{M}l_{\Sigma}^{\square\exists *(m)}\left(N_m,\ M;\ w\right)} \tag{73}$$

reflects a speed gain of the sequential parallelization by using the rectangular cell clustering pattern.

Furthermore, the rectangular cell clustering pattern can be compared with the iterative clustering by (16) — (22). Then the approximately shortest route is still assembled in accordance with (62) — (67), where centroids (23) are used (e. g., see Figure 8) instead of the rectangular cell centers. For this case, the two successive subtours for open-loop TSP u_m and open-loop TSP u_{m+1}, $m = \overline{1,\ M-1}$, are connected by a node from cluster u_m which is the closest to cluster u_{m+1} and simultaneously is the farthest from the depot. Inasmuch this problem usually does not have a solution, a node from the respective Pareto set is selected [35, 13]. Here, denote the shortest length of the subroute in the open-loop TSP for cluster m by $\tilde{\rho}_{\Sigma}^{\exists *(m)}\left(N_m,\ M;\ w\right)$, and denote by $l_{\Sigma}^{\exists *(m)}\left(N_m,\ M;\ w\right)$ the number of iterations taken to solve this open-loop TSP. Then a ratio

$$f\left(N,\ M;\ w\right) = \frac{\sum\limits_{m=1}^{M}\tilde{\rho}_{\Sigma}^{\exists *(m)}\left(N_m,\ M;\ w\right)}{\sum\limits_{m=1}^{M}\tilde{\rho}_{\Sigma}^{\square\exists *(m)}\left(N_m,\ M;\ w\right)} \tag{74}$$

reflects an accuracy gain of the parallelization with respect to the iterative clustering by using the rectangular assembling approach. A ratio

$$\mu\left(N,\ M;\ w\right) = \frac{\sum\limits_{m=1}^{M}l_{\Sigma}^{\exists *(m)}\left(N_m,\ M;\ w\right)}{\sum\limits_{m=1}^{M}l_{\Sigma}^{\square\exists *(m)}\left(N_m,\ M;\ w\right)} \tag{75}$$

reflects a speed gain of the sequential parallelization with respect to the iterative clustering by using the rectangular assembling approach, without taking into account the time spent on the iterative clustering.

Another approach is to build an assembling polyline by solving a supplementary TSP, in which the nodes are centroids (23). It is called the centroid TSP taking $l_{\Sigma}^{(\mathbf{C})*}\left(M;\ w\right)$ iterations to obtain a solution. For the case of the centroid TSP, denote the shortest length of the subroute in the subproblem for cluster m by $\tilde{\rho}_{\Sigma}^{(\mathbf{C})*(m)}\left(N_m,\ M;\ w\right)$. Then a ratio

$$c\left(N,\ M;\ w\right) = \frac{\sum\limits_{m=1}^{M}\tilde{\rho}_{\Sigma}^{(\mathbf{C})*(m)}\left(N_m,\ M;\ w\right)}{\sum\limits_{m=1}^{M}\tilde{\rho}_{\Sigma}^{\square\exists *(m)}\left(N_m,\ M;\ w\right)} \tag{76}$$

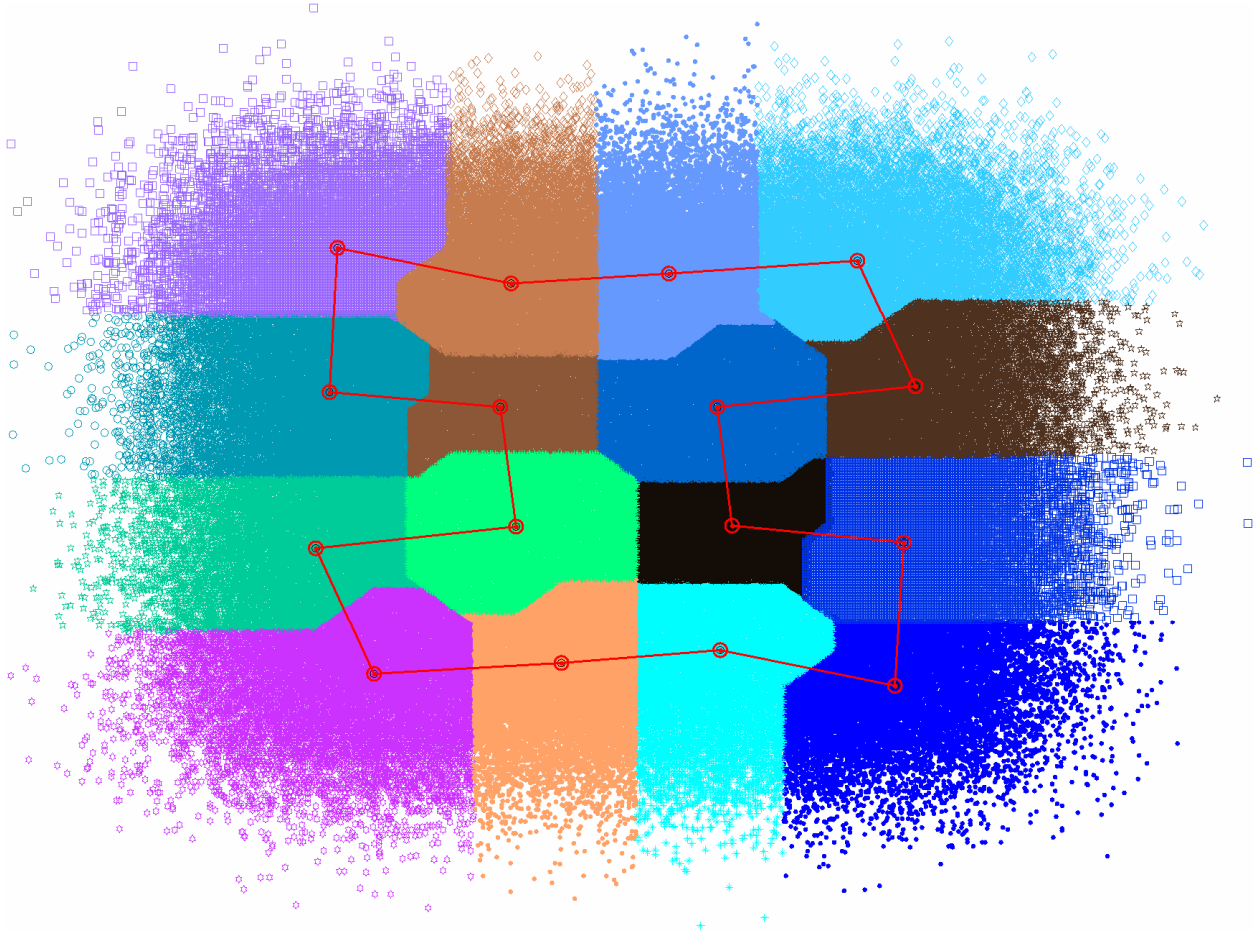

Figure 8. The set of 250000 nodes from Figure 1 (the depot is not marked), where the approximately shortest route is assembled in accordance with (62) — (67) passing through the 16 centroids of the clusters

reflects an accuracy gain of the parallelization with respect to the iterative clustering by using the centroid TSP assembling approach. A ratio

$$\gamma\left(N,\ M;\ w\right)=\frac{l_{\Sigma}^{*(\mathbf{C})}\left(M;\ w\right)+\sum\limits_{m=1}^{M}l_{\Sigma}^{(\mathbf{C})*(m)}\left(N_m,\ M;\ w\right)}{\sum\limits_{m=1}^{M}l_{\Sigma}^{\square\exists*(m)}\left(N_m,\ M;\ w\right)}, \tag{77}$$

where $l_{\Sigma}^{(\mathbf{C})*(m)}\left(N_m,\ M;\ w\right)$ is the number of iterations taken to solve centroid-based subproblem m, reflects a speed gain of the sequential parallelization with respect to the iterative clustering by using the centroid TSP assembling approach, without taking into account the time spent on the iterative clustering.

Each of ratios (73), (75), (77) implies that the open-loop TSPs are not solved in parallel, though. When they are solved in parallel, then the speed gains are:

$$\lambda^{(\text{par})}\left(N,\ M;\ w\right)=\frac{l_{\Sigma}^{*}\left(N;\ w\right)}{\max\limits_{m=\overline{1,\ M}}l_{\Sigma}^{\square\exists*(m)}\left(N_m,\ M;\ w\right)}, \tag{78}$$

$$\mu^{(\text{par})}(N,\ M;\ w) = \frac{\max\limits_{m=\overline{1,\ M}} l_{\Sigma}^{\exists *(m)}(N_m,\ M;\ w)}{\max\limits_{m=\overline{1,\ M}} l_{\Sigma}^{\square\exists *(m)}(N_m,\ M;\ w)}, \tag{79}$$

$$\gamma^{(\text{par})}(N,\ M;\ w) = \frac{l_{\Sigma}^{*(\mathbf{C})}(M;\ w) + \max\limits_{m=\overline{1,\ M}} l_{\Sigma}^{(\mathbf{C})*(m)}(N_m,\ M;\ w)}{\max\limits_{m=\overline{1,\ M}} l_{\Sigma}^{\square\exists *(m)}(N_m,\ M;\ w)}, \tag{80}$$

by the supposition of that the open-loop TSPs are simultaneously solved on M parallel processor cores. In contrast to gains (78) — (80), gains (73), (75), (77) suppose that the TSPs are solved on a single processor core.

If $g(N,\ M;\ w) > 1$ then it means that the length of the route assembled from the rectangular-cell-clustering-based subroutes is shorter than that length $\tilde{\rho}_{\Sigma}^{*}(N;\ w)$ obtained without the parallelization. If $g(N,\ M;\ w) < 1$ then the parallelization worsens the accuracy of the given TSP solution. If $f(N,\ M;\ w) > 1$ then it means that the length of the route assembled from the rectangular-cell-clustering-based subroutes is shorter than the length of the route assembled from the iterative-clustering-based subroutes. If $f(N,\ M;\ w) < 1$ then the rectangular-cell-clustering-based parallelization is worse than the iterative-clustering-based one for the given TSP. However, in the case $f(N,\ M;\ w) = 1$ the rectangular-cell-clustering-based parallelization still works because the iterative-clustering-based parallelization additionally spends computational time to cluster nodes (1) iteratively rather than to apply the rectangular cell clustering pattern. If $c(N,\ M;\ w) > 1$ then it means that the length of the route assembled from the rectangular-cell-clustering-based subroutes is shorter than that length assembled from the centroid-based subroutes. If $c(N,\ M;\ w) < 1$ then the rectangular-cell-clustering-based parallelization is worse than the centroid-based one for the given TSP. However, in the case $c(N,\ M;\ w) = 1$ the rectangular-cell-clustering-based parallelization still works because the centroid-based parallelization additionally spends computational time to cluster nodes (1) iteratively, whereupon the centroid TSP is additionally solved before assembling the subroutes.

If $\lambda(N,\ M;\ w) > 1$ then it means that, even when the open-loop TSPs are solved sequentially, the route assembled from the rectangular-cell-clustering-based subroutes is obtained faster than a solution without the parallelization. If $\lambda^{(\text{par})}(N,\ M;\ w) > 1$ then it means that the rectangular-cell-clustering-based parallelization works faster by solving the subproblems in parallel. If $\mu(N,\ M;\ w) > 1$ then it means that, even when the subproblems are solved sequentially, the rectangular-cell-clustering-based parallelization is faster than the iterative-clustering-based one for the given TSP; if $\mu^{(\text{par})}(N,\ M;\ w) > 1$ then it means that the rectangular-cell-clustering-based parallelization is faster than the iterative-clustering-based one when the subproblems are solved in parallel. Speed gains (77) and (80) are treated similarly.

To study the statistics of the parallelization gains by (72) — (80), their minimal, average, and maximal values calculated respectively as

$$g_{\min}(N,\ M) = \min_{w=\overline{1,\ 30}} g(N,\ M;\ w),\quad g_{\max}(N,\ M) = \max_{w=\overline{1,\ 30}} g(N,\ M;\ w), \tag{81}$$

$$\bar{g}(N,\ M) = \frac{1}{30}\cdot\sum_{w=1}^{30} g(N,\ M;\ w), \tag{82}$$

$$f_{\min}(N,\ M) = \min_{w=\overline{1,\ 30}} f(N,\ M;\ w),\quad f_{\max}(N,\ M) = \max_{w=\overline{1,\ 30}} f(N,\ M;\ w), \tag{83}$$

$$c_{\min}(N,\ M) = \min_{w=\overline{1,\ 30}} c(N,\ M;\ w),\quad c_{\max}(N,\ M) = \max_{w=\overline{1,\ 30}} c(N,\ M;\ w), \tag{84}$$

$$\bar{f}(N,\ M) = \frac{1}{30}\cdot\sum_{w=1}^{30} f(N,\ M;\ w),\quad \bar{c}(N,\ M) = \frac{1}{30}\cdot\sum_{w=1}^{30} c(N,\ M;\ w), \tag{85}$$

$$\lambda_{\min}(N,\ M) = \min_{w=\overline{1,\ 30}} \lambda(N,\ M;\ w),\quad \lambda_{\max}(N,\ M) = \max_{w=\overline{1,\ 30}} \lambda(N,\ M;\ w), \tag{86}$$

$$\bar{\lambda}\,(N,\ M) = \frac{1}{30}\cdot\sum_{w=1}^{30}\lambda\,(N,\ M;\ w), \tag{87}$$

$$\mu_{\min}\,(N,\ M) = \min_{w=\overline{1,\ 30}}\mu\,(N,\ M;\ w)\,,\quad \mu_{\max}\,(N,\ M) = \max_{w=\overline{1,\ 30}}\mu\,(N,\ M;\ w)\,, \tag{88}$$

$$\gamma_{\min}\,(N,\ M) = \min_{w=\overline{1,\ 30}}\gamma\,(N,\ M;\ w)\,,\quad \gamma_{\max}\,(N,\ M) = \max_{w=\overline{1,\ 30}}\gamma\,(N,\ M;\ w)\,, \tag{89}$$

$$\bar{\mu}\,(N,\ M) = \frac{1}{30}\cdot\sum_{w=1}^{30}\mu\,(N,\ M;\ w),\quad \bar{\gamma}\,(N,\ M) = \frac{1}{30}\cdot\sum_{w=1}^{30}\gamma\,(N,\ M;\ w), \tag{90}$$

$$\lambda^{(\mathrm{par})}_{\min}\,(N,\ M) = \min_{w=\overline{1,\ 30}}\lambda^{(\mathrm{par})}\,(N,\ M;\ w)\,,\quad \lambda^{(\mathrm{par})}_{\max}\,(N,\ M) = \max_{w=\overline{1,\ 30}}\lambda^{(\mathrm{par})}\,(N,\ M;\ w)\,, \tag{91}$$

$$\bar{\lambda}^{(\mathrm{par})}\,(N,\ M) = \frac{1}{30}\cdot\sum_{w=1}^{30}\lambda^{(\mathrm{par})}\,(N,\ M;\ w), \tag{92}$$

$$\mu^{(\mathrm{par})}_{\min}\,(N,\ M) = \min_{w=\overline{1,\ 30}}\mu^{(\mathrm{par})}\,(N,\ M;\ w)\,,\quad \mu^{(\mathrm{par})}_{\max}\,(N,\ M) = \max_{w=\overline{1,\ 30}}\mu^{(\mathrm{par})}\,(N,\ M;\ w)\,, \tag{93}$$

$$\gamma^{(\mathrm{par})}_{\min}\,(N,\ M) = \min_{w=\overline{1,\ 30}}\gamma^{(\mathrm{par})}\,(N,\ M;\ w)\,,\quad \gamma^{(\mathrm{par})}_{\max}\,(N,\ M) = \max_{w=\overline{1,\ 30}}\gamma^{(\mathrm{par})}\,(N,\ M;\ w)\,, \tag{94}$$

$$\bar{\mu}^{(\mathrm{par})}\,(N,\ M) = \frac{1}{30}\cdot\sum_{w=1}^{30}\mu^{(\mathrm{par})}\,(N,\ M;\ w),\quad \bar{\gamma}^{(\mathrm{par})}\,(N,\ M) = \frac{1}{30}\cdot\sum_{w=1}^{30}\gamma^{(\mathrm{par})}\,(N,\ M;\ w) \tag{95}$$

are to be considered. The statistics presented in Table 1 shows that parallelizing the TSPs by the rectangular cell clustering is at least four times faster than solving the whole TSP. The accuracy losses by four clusters ($M = 4$) do not exceed 1.5 % on average, where the worst case is

$$g_{\min}\,(2001,\ 4) = 0.9711$$

which is a 2.89 % accuracy loss. As the number of clusters is increased, the accuracy loss slowly grows. The growth is slower as the TSP size increases. The speedups by (73), (78), (86), (87), (91), (92) are greater at a fewer nodes to be clustered. These gains are obviously increasing as the number of clusters is increased. Overall, the average speedup by solving on a single processor core varies between 4.539 and 25.6082. If four cores are used, it varies between 16.5919 and 19.3173. By using 16 cores, the average speedup drops from 96.7264 at 2001 nodes down to 51.8779 at 8001 nodes. Nevertheless, it does not have a distinct drop trend when 64 TSPs are simultaneously solved on 64 cores, varying between 486.3396 and 666.0474.

Table 2 shows the statistics with respect to the iterative clustering by (16)—(22), where centroids (23) are used to assemble a closed-loop route instead of using the rectangular cell centers. Based on comparing to the rectangular cell clustering pattern, accuracy gain (74) bounces above 1 (it is highlighted bold) and below. As the number of clusters is increased and as the TSP size increases, the assembling by the rectangular cell clustering becomes more accurate reaching a 4.45 % accuracy gain at 4001 nodes and 64 clusters. The worst case is at 2001 nodes and 16 clusters, where an accuracy loss is about 2.5 %. The speedups by (75), (79), (88), $\bar{\mu}\,(N,\ M)$ by (90), (93), $\bar{\mu}^{(\mathrm{par})}\,(N,\ M)$ by (95) are badly scattered around 1 (when both the iterative clustering and rectangular cell clustering have the same computational speed). The rectangular cell clustering is 1.12 % to 24.79 % faster at the fewest clusters ($M = 4$), when the four open-loop TSPs are simultaneously solved on four cores. Nevertheless, it is twice as faster at 64 clusters and 6001 nodes. Contrary to that, there is a TSP with 4001 nodes divided into 16 open-loop TSPs, where the iterative clustering is more than 55 % faster on 16 cores. Highlighted bold, the speedups by the rectangular cell clustering have a vague trend, especially when TSPs are solved on a single processor core. Solving on M cores seems more preferable for the rectangular cell clustering, where using four cores is always faster.

Table 1. Statistics of accuracy gain (72) and speed gains (73), (78) in parallelizing the TSPs generated by (68) — (71)

N	2001			4001			6001			8001		
M	4	16	64	4	16	64	4	16	64	4	16	64
$g_{\min}(N,\ M)$	0.9711	0.9429	0.9384	0.9831	0.9668	0.9504	0.9863	0.9689	0.9582	0.9789	0.9768	0.9637
$\bar{g}(N,\ M)$	0.9851	0.9627	0.9479	0.9903	0.9755	0.958	0.988	0.9775	0.9609	0.9877	0.9829	0.9717
$g_{\max}(N,\ M)$	0.9976	0.9699	0.9556	1.0086	0.9857	0.9743	0.9896	0.9837	0.9647	0.9949	0.9933	0.9768
$\lambda_{\min}(N,\ M)$	4.119	9.6594	15.7502	3.976	11.5658	23.284	5	9.7901	24.2954	5	7.9657	19.8396
$\bar{\lambda}(N,\ M)$	5.1358	12.7678	20.6329	4.539	12.4419	25.6082	5.0002	10.6553	24.8371	5.0003	8.2547	20.317
$\lambda_{\max}(N,\ M)$	6.1098	14.608	23.1692	5.248	13.8607	27.0506	5.0008	10.9908	25.6865	5.0006	8.4995	20.6805
$\lambda_{\min}^{(\text{par})}(N,\ M)$	13.8837	61.3343	387.1463	14.3559	70.6297	601.2988	18.9994	59.4183	585.6515	19.0386	45.7143	425.8264
$\bar{\lambda}^{(\text{par})}(N,\ M)$	17.9943	96.7264	612.3673	16.5919	83.1698	666.0474	19.3173	67.2631	634.6565	19.288	51.8779	486.3396
$\lambda_{\max}^{(\text{par})}(N,\ M)$	22.7247	118.4044	710.1305	19.3988	96.4761	763.757	19.6335	74.2841	664.5254	19.4458	60.8657	565.931

Table 2. Statistics of accuracy gain (74) and speed gains (75), (79) in parallelizing the TSPs generated by (68) — (71)

N	2001			4001			6001			8001		
M	4	16	64	4	16	64	4	16	64	4	16	64
$f_{\min}(N,\ M)$	0.977	0.9749	0.9921	0.9933	0.9899	0.9999	0.9922	0.9873	**1.0033**	0.9909	0.9906	**1.0062**
$\bar{f}(N,\ M)$	0.9961	0.9874	**1.0147**	**1.0014**	0.9968	**1.0247**	**1.001**	0.9977	**1.0148**	**1.0007**	0.9978	**1.0131**
$f_{\max}(N,\ M)$	**1.0134**	0.9962	**1.034**	**1.0153**	**1.0064**	**1.0445**	**1.0067**	**1.0063**	**1.0221**	**1.0112**	**1.0076**	**1.0252**
$\mu_{\min}(N,\ M)$	0.9359	0.9077	0.9521	0.9337	0.81	0.8813	1	0.8863	0.8733	1	0.8388	0.7981
$\bar{\mu}(N,\ M)$	**1.0184**	0.9484	0.9726	0.9815	0.9186	0.9077	1	0.9217	0.8977	**1.0001**	0.8803	0.8314
$\mu_{\max}(N,\ M)$	**1.1386**	0.9938	0.9891	**1.0415**	**1.0312**	0.945	**1.0002**	0.9714	0.9215	**1.0001**	0.9288	0.856
$\mu_{\min}^{(\text{par})}(N,\ M)$	**1.0329**	0.6032	0.776	**1.0164**	0.4446	0.7294	**1.0641**	0.5226	0.7229	**1.0112**	0.5244	0.5776
$\bar{\mu}^{(\text{par})}(N,\ M)$	**1.1436**	0.7145	**1.0056**	**1.0827**	0.6529	0.8669	**1.0882**	0.6344	0.9798	**1.0525**	0.6088	0.7098
$\mu_{\max}^{(\text{par})}(N,\ M)$	**1.2479**	0.8426	**1.139**	**1.2237**	0.9274	0.9272	**1.1284**	0.7032	**1.5033**	**1.1011**	0.7248	0.8321

Computational speed compared to the centroid TSP assembling approach (Table 3) does not have a distinct trend either. The speedups by (77), (80), (89), $\bar{\gamma}(N,\ M)$ by (90), (94), $\bar{\gamma}^{(\text{par})}(N,\ M)$ by (95) are badly scattered around 1 also, although solving on M cores seems more preferable for the rectangular cell clustering (there are more speedups highlighted bold in the bottom three lines corresponding to solving on M parallel processor cores). Using four cores is almost always faster, with two exceptions at 2001 and 4001 nodes. Accuracy gain (76) bouncing above 1 (it is highlighted bold) and below shows that the rectangular cell clustering is more accurate on average, with four exceptions at 2001 nodes and $M \in \{4,\ 16\}$, at 6001 nodes and 16 clusters, and at 8001 nodes and 16 clusters.

A visual example of solving a whole TSP generated for 8001 nodes is presented in Figure 9, where the route length is 13035.4899. It is seen that the density of nodes closer to the margins is lesser. This is reasoned by partially using the normal distribution in generating nodes by (70) and (71). The rectangular cell clustering thus produces too scattered cluster sizes. By dividing the exemplified TSP into 64 clusters, the cluster size varies between 6 and

Table 3. Statistics of accuracy gain (76) and speed gains (77), (80) in parallelizing the TSPs generated by (68) — (71)

N	2001			4001			6001			8001		
M	4	16	64	4	16	64	4	16	64	4	16	64
$c_{\min}(N,\ M)$	0.9851	0.9694	0.9954	0.9944	0.997	**1.0027**	0.9978	0.9892	0.9964	0.998	0.9892	**1.0036**
$\bar{c}(N,\ M)$	0.9987	0.9906	**1.0117**	**1.0038**	**1.0062**	**1.0111**	**1.0002**	0.9963	**1.0053**	**1.0016**	0.9976	**1.0082**
$c_{\max}(N,\ M)$	**1.0098**	**1.0053**	**1.0242**	**1.0141**	**1.0158**	**1.0247**	**1.0028**	**1.0018**	**1.0127**	**1.0105**	**1.0134**	**1.0129**
$\gamma_{\min}(N,\ M)$	0.9794	0.9136	0.9797	0.8114	0.8591	0.8951	0.9785	0.8387	0.8647	**1.0001**	0.8317	0.8073
$\bar{\gamma}(N,\ M)$	**1.0395**	0.9546	0.9978	0.9479	0.919	0.9223	0.9958	0.9135	0.8895	**1.0001**	0.8811	0.8256
$\gamma_{\max}(N,\ M)$	**1.1008**	**1.0123**	**1.0035**	**1.0204**	0.9731	0.9458	**1.0002**	0.9478	0.919	**1.0002**	0.9069	0.8427
$\gamma_{\min}^{(\text{par})}(N,\ M)$	0.9222	0.5993	**1.6982**	0.9584	0.557	**1.1509**	**1.0644**	0.595	0.9416	**1.0114**	0.4892	0.6924
$\bar{\gamma}^{(\text{par})}(N,\ M)$	**1.1792**	0.7992	**2.0086**	**1.0674**	0.6828	**1.2344**	**1.0885**	0.7127	**1.0626**	**1.0527**	0.5912	0.8091
$\gamma_{\max}^{(\text{par})}(N,\ M)$	**1.355**	**1.0226**	**2.3167**	**1.2242**	**1.0445**	**1.3001**	**1.1286**	0.8321	**1.3918**	**1.1013**	0.6243	0.9425

256 nodes. This results in an aggregated route whose length is 13363.2911 (see Figure 10, where the cluster of 6 nodes is at the top right corner) being just 2.51 % longer than that in Figure 9. However, the approximated solution in Figure 10 is obtained 20.4818 times faster if the 64 open-loop TSPs are solved on a single processor core. Furthermore, it is obtained 425.8264 times faster if every open-loop TSP is solved on its own processor core — it is the worst case wherein

$$\lambda_{\min}^{(\text{par})}(8001,\ 64) = 425.8264$$

in Table 1.

When the TSP in Figure 9 is solved by using the iterative clustering by (16) — (22), where an approximately shortest route is assembled in accordance with (62) — (67) passing through 64 centroids (23), the results become better in speedup but worse in accuracy. Whereas the speed gains (75), (79) here are

$$\mu(8001,\ 64;\ 5) = 0.8451$$

and

$$\mu^{(\text{par})}(8001,\ 64;\ 5) = 0.6299,$$

the respective assembled route (Figure 11) turns out to be 1.54 % longer than that in Figure 10. The huge drop in the rectangular cell clustering speedup is reasoned by that the cluster size upon the iterative clustering varies between 45 and 210 nodes, which is a significantly narrower range compared to that for the rectangular cell clustering.

Using the centroid TSP assembling approach results in similar rectangular cell clustering speedup drops and accuracy gain:

$$\gamma(8001,\ 64;\ 5) = 0.8427,$$

$$\gamma^{(\text{par})}(8001,\ 64;\ 5) = 0.7201,$$

$$c(8001,\ 64;\ 5) = 1.0121.$$

The assembled closed-loop route shown in Figure 12 is slightly shorter than that in Figure 11; its length is 13525.3716. It is also worth noting that the set of nodes $\{k_m^{***}\}_{m=1}^{63}$ connecting the open-loop subroutes differs from the analogous set in Figure 11, although the clusters are the same.

It may seem that the rectangular cell clustering pattern does not have a clear advantage over the iterative clustering, whether a route is assembled by the centroid TSP assembling approach or by the rectangular assembling approach. However, as the portion of the normal distribution in generating nodes by (70) and (71) is reduced, the

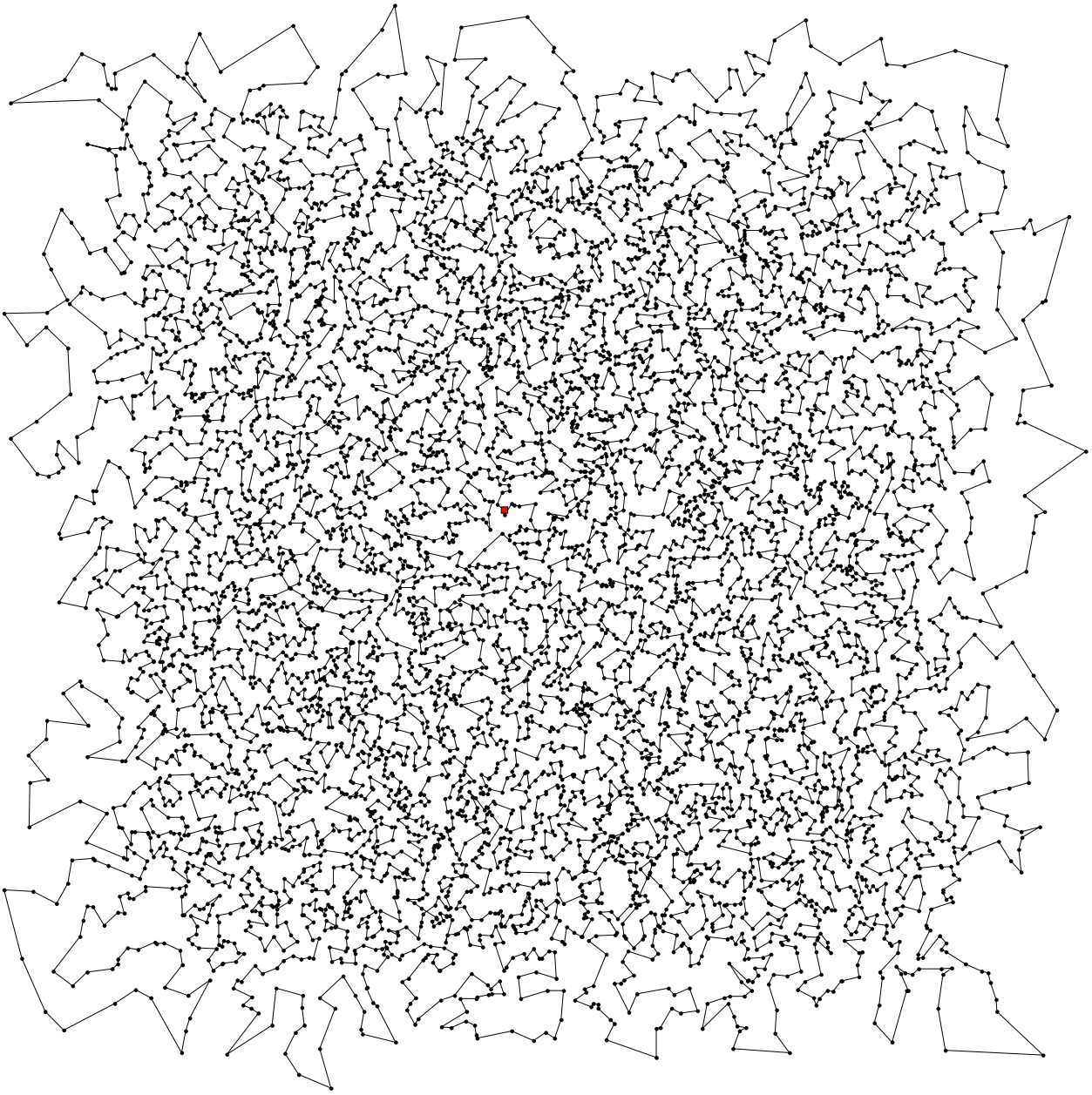

Figure 9. An approximately shortest route for a TSP with 8001 nodes solved without clustering and parallelization

rectangular cell clustering approach becomes more accurate and faster. Thus, if the node locations are generated as

$$p_{k1} = 150 \cdot \theta_1 + 2 \cdot \eta_1 + 50 \;\text{ and }\; p_{k2} = 150 \cdot \theta_2 + 2 \cdot \eta_2 + 50 \;\text{ by }\; k = \overline{2,\ N} \tag{96}$$

instead of (70), then it is 0.13 % more accurate and 1.02 % faster than the rectangular assembling approach (Table 4). Compared to the latter, the rectangular cell clustering approach is 26.42 % faster also when M parallel processor cores are used. In addition, it is still 0.06 % more accurate than the centroid TSP assembling approach being 2.61 % slower on a single processor core and 52.22 % slower on M parallel processor cores (Table 5).

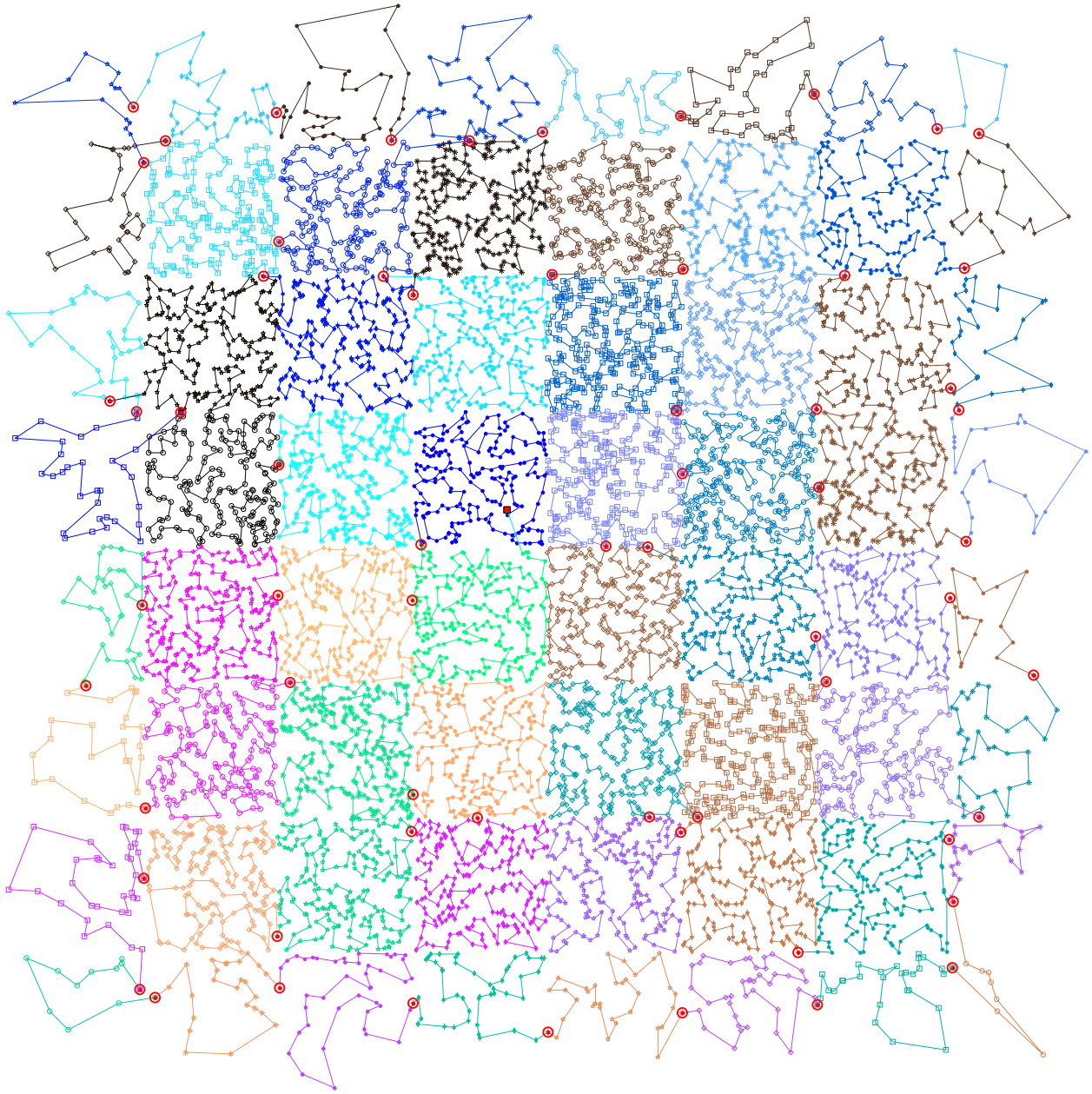

Figure 10. An approximately shortest route for the TSP in Figure 9 solved by the rectangular cell clustering pattern; the route, whose length is 13363.2911, is assembled in accordance with (62) — (67) passing through the 64 numbered centers of the lattice cells, where nodes $\{k_m^{***}\}_{m=1}^{63}$ connecting the open-loop subroutes are marked as circles

Measuring computational time in seconds, the rectangular cell clustering approach is 67.16 % and 59.03 % faster than the rectangular assembling and centroid TSP assembling approaches, respectively. This time includes an amount of time spent on the clustering itself, i. e. on the preparation to solve. As it has been mentioned above, the rectangular cell clustering is very fast, so it is no wonder that the preparation time of the rectangular cell clustering approach is 60000 to 70000 times shorter than preparing to solve by the other two approaches. If to consider only "pure" computational time spent on solving the subproblems, without the preparation time, then the rectangular

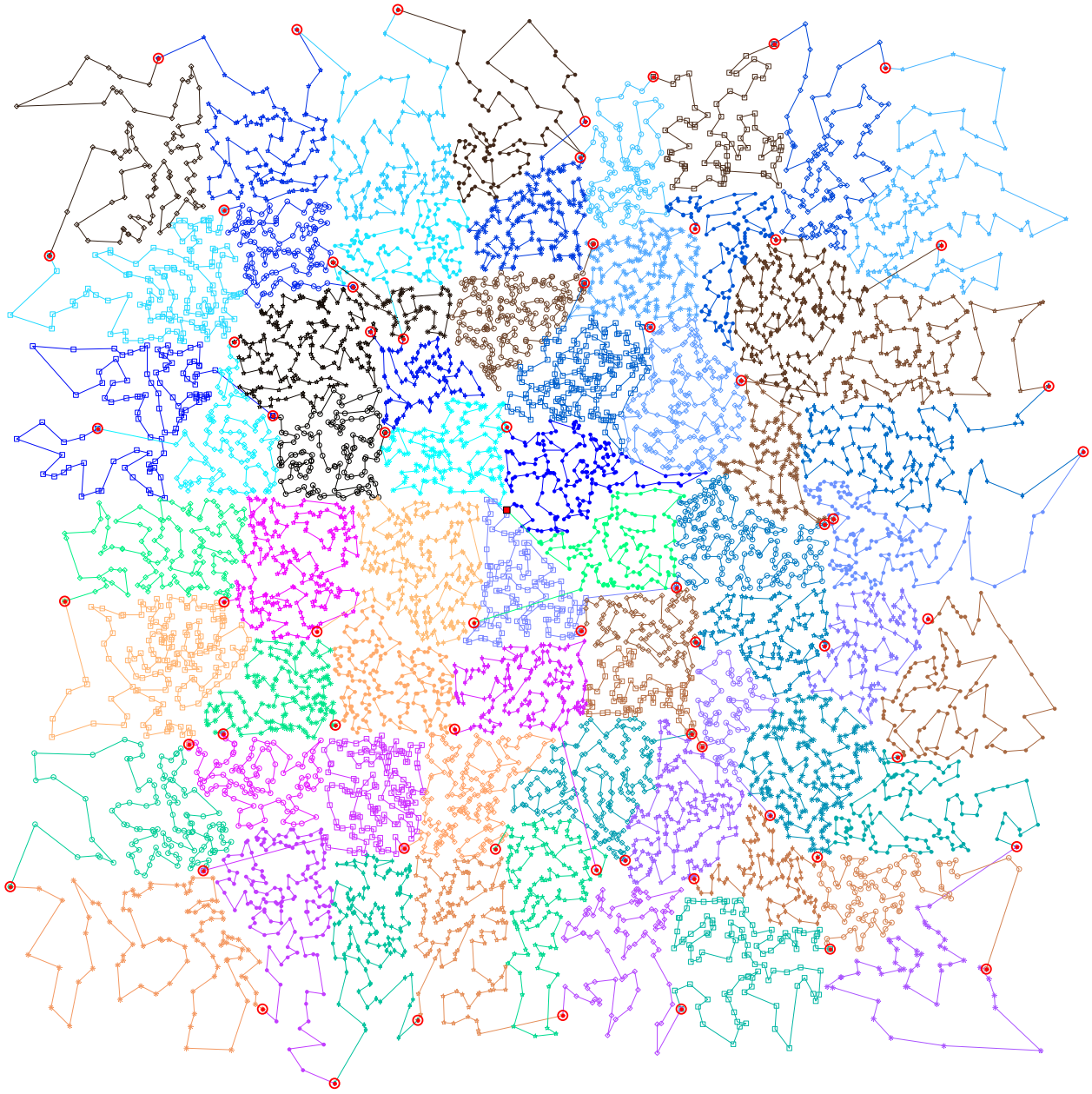

Figure 11. An approximately shortest route for the TSP in Figure 9 solved by the iterative clustering by (16) — (22), where centroids (23) are used to assemble a closed-loop route instead of using the rectangular cell centers in Figure 10; the route length is 13569.1552 which is 1.54 % longer than that in Figure 10

cell clustering approach is 2.43 % and 1 % faster than the rectangular assembling and centroid TSP assembling approaches, respectively. As the number of clusters is increased, the advantage strengthens with respect to the rectangular assembling, and it slightly weakens with respect to the centroid TSP assembling approach. As the TSP size increases, the advantage strengthens with respect to them both following a quadratic pattern.

It is quite obvious that numbering the cell centers by (62) — (67) is not the only possible version. Before the assembling, the clusters are re-numbered so that their new numbers correspond to the consecution of how the clusters are connected by the symmetric rectangular closed-loop serpentine. In particular, the cluster containing the

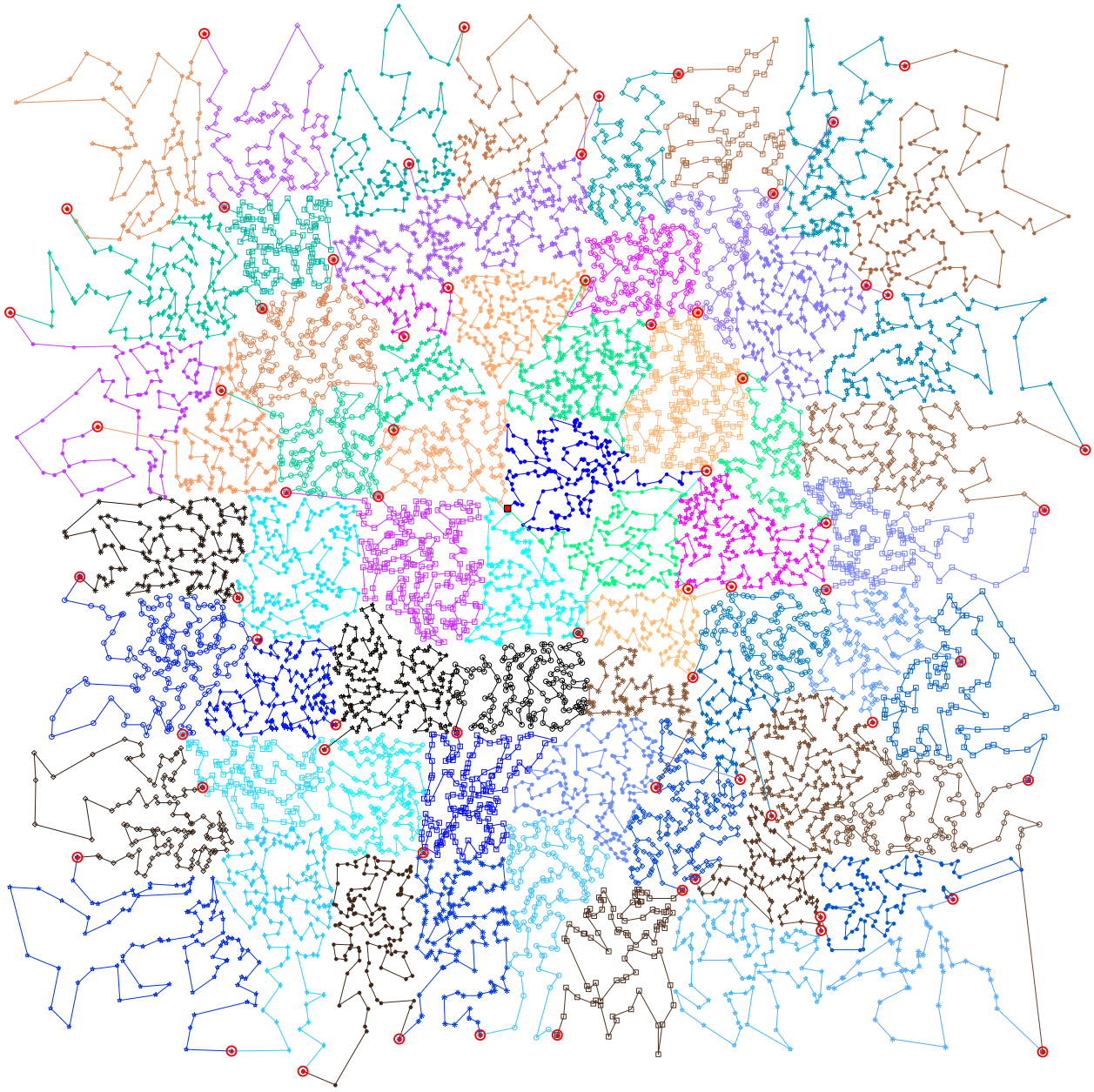

Figure 12. The assembled closed-loop route for the TSP in Figure 9 solved by the centroid TSP assembling approach

depot is always re-numbered so that its number is 1. The same is done for the polyline closed-loop serpentine (see Figure 8), where the cluster closest to every cell center is determined.

Connecting successive subtours for open-loop TSPs in the case of iterative clustering (when either the rectangular assembling or centroid TSP assembling approach is used) is fulfilled by determining connectors $\{k_m^{***}\}_{m=1}^{M-1}$, where node k_m^{***} belonging to cluster m (after re-numbering) is the destination node in open-loop TSP m. The assembling is done almost trivially for just two clusters (19) and (20). The salesman departing from the depot must complete the open-loop subtour at a node of cluster (19) that is the farthest from the depot. On the other hand, this node must be the nearest to cluster (20) to resume building the route for the initial TSP via the shortest connection of the two

Table 4. Statistics of accuracy gain (74) and speed gains (75), (79) in parallelizing the TSPs generated by (68), (69), (96), (71)

N	2001			4001			6001			8001		
M	4	16	64	4	16	64	4	16	64	4	16	64
$f_{\min}(N,\ M)$	0.9817	0.991	**1.012**	0.9885	0.9904	**1.0052**	0.9965	0.984	0.9981	0.9936	0.9983	**1.0071**
$\bar{f}(N,\ M)$	0.9894	0.9944	**1.021**	0.9951	0.994	**1.0131**	**1.0006**	0.9969	**1.0024**	0.9963	**1.0003**	**1.0124**
$f_{\max}(N,\ M)$	**1.0001**	**1.0004**	**1.033**	**1.0019**	0.9985	**1.0171**	**1.0036**	**1.0098**	**1.0062**	**1.0015**	**1.0013**	**1.0156**
$\mu_{\min}(N,\ M)$	0.8557	0.9232	1.0036	0.9617	0.9903	0.9978	1	0.9751	0.9954	1	0.9794	**1.028**
$\bar{\mu}(N,\ M)$	0.9843	0.9723	**1.0162**	**1.0188**	**1.0238**	**1.0221**	1	0.9874	**1.0104**	1	**1.0225**	**1.064**
$\mu_{\max}(N,\ M)$	**1.1214**	**1.0258**	**1.0245**	**1.0871**	**1.0785**	**1.0382**	1	**1.0101**	**1.0218**	**1.0001**	**1.0592**	**1.0863**
$\mu_{\min}^{(\text{par})}(N,\ M)$	0.9806	0.8472	**1.2724**	0.9459	**1.1808**	**1.2554**	**1.0621**	0.9055	**1.1423**	**1.0819**	**1.2971**	**1.3505**
$\bar{\mu}^{(\text{par})}(N,\ M)$	**1.1901**	**1.0587**	**1.3368**	**1.0449**	**1.2731**	**1.3612**	**1.0731**	**1.126**	**1.4594**	**1.1017**	**1.5179**	**1.6272**
$\mu_{\max}^{(\text{par})}(N,\ M)$	**1.4493**	**1.2966**	**1.3866**	**1.1512**	**1.3358**	**1.4952**	**1.0858**	**1.3131**	**1.9044**	**1.1174**	**1.772**	**1.8874**

Table 5. Statistics of accuracy gain (76) and speed gains (77), (80) in parallelizing the TSPs generated by (68), (69), (96), (71)

N	2001			4001			6001			8001		
M	4	16	64	4	16	64	4	16	64	4	16	64
$c_{\min}(N,\ M)$	0.9853	0.9911	0.9804	0.9873	0.9899	0.9981	0.994	0.9859	**1.0043**	0.9962	0.9938	**1.0032**
$\bar{c}(N,\ M)$	0.9913	0.9953	**1.005**	0.9954	0.998	**1.0083**	0.9994	0.9935	**1.018**	0.9984	0.9966	**1.0082**
$c_{\max}(N,\ M)$	0.9977	0.9986	**1.0206**	**1.0056**	**1.0097**	**1.0144**	**1.0066**	**1.0037**	**1.028**	**1.0002**	0.9984	**1.0169**
$\gamma_{\min}(N,\ M)$	0.8276	0.9644	**1.0627**	0.9426	0.9604	**1.0122**	**1.0001**	**1.0354**	0.9975	**1.0001**	**1.042**	**1.0192**
$\bar{\gamma}(N,\ M)$	0.9931	0.9908	**1.0649**	0.9847	**1.0232**	**1.0401**	**1.0001**	**1.0719**	**1.0179**	**1.0001**	**1.0688**	**1.0581**
$\gamma_{\max}(N,\ M)$	**1.1538**	**1.0339**	**1.0682**	**1.027**	**1.0786**	**1.0668**	**1.0001**	**1.1273**	**1.0353**	**1.0002**	**1.1011**	**1.0817**
$\gamma_{\min}^{(\text{par})}(N,\ M)$	**1.0011**	0.9851	**2.5909**	0.9747	**1.1212**	**1.9759**	**1.0624**	**1.03**	**1.4156**	**1.0821**	**1.3855**	**1.6291**
$\bar{\gamma}^{(\text{par})}(N,\ M)$	**1.1773**	**1.167**	**2.7244**	**1.0341**	**1.3865**	**2.1456**	**1.0734**	**1.325**	**1.6587**	**1.1019**	**1.5775**	**1.8955**
$\gamma_{\max}^{(\text{par})}(N,\ M)$	**1.4897**	**1.3657**	**2.8386**	**1.134**	**1.5736**	**2.4166**	**1.0861**	**1.5694**	**2.0273**	**1.1176**	**1.6896**	**2.1112**

open-loop subtours. Therefore, the first open-loop subtour destination node must have number

$$k_1^* \in \arg \max_{k_1 \in K_1(1)} \rho(k_1,\ 1) \tag{97}$$

but this number also should be $k_1^{**} \in K_1(1)$ such that

$$\{k_1^{**},\ k_2^{**}\} \in \arg \min_{k_1 \in K_1(1),\ k_2 \in K_2(1)} \rho(k_1,\ k_2)\,. \tag{98}$$

Although the case $k_1^* = k_1^{**}$ is not impossible, it is rather unlikely. So, in the case $k_1^* \neq k_1^{**}$, a node $k_1^{***} \in K_1(1)$ that is the best Pareto-efficient point is selected. For this, distances $\rho(k_1,\ 1)$ and $\delta(k_1,\ k_2)$, where

$$\delta(k_1,\ k_2) = \max\{\rho(k_1,\ 1),\ \rho(k_1,\ k_2)\} - \rho(k_1,\ k_2)\,, \tag{99}$$

are considered as a two-component vector

$$\mathbf{V}\left(k_1,\ k_2\right)=\left[\ \rho\left(k_1,\ 1\right)\quad \delta\left(k_1,\ k_2\right)\ \right]. \tag{100}$$

A subset

$$\tilde{V}\subset V=\left\{\mathbf{V}\left(k_1,\ k_2\right):\ k_1\in K_1\left(1\right),\ k_2\in K_2\left(1\right)\right\} \tag{101}$$

is selected such that for every $\tilde{\mathbf{V}}\left(\tilde{k}_1,\ \tilde{k}_2\right)\in\tilde{V}$ and every $\mathbf{V}_0\left(k_1^{(0)},\ k_2^{(0)}\right)\in\left\{V\backslash\tilde{V}\right\}$ either a pair of inequalities

$$\rho\left(\tilde{k}_1,\ 1\right)\geqslant\rho\left(k_1^{(0)},\ 1\right)\ \text{ and }\ \delta\left(\tilde{k}_1,\ \tilde{k}_2\right)>\delta\left(k_1^{(0)},\ k_2^{(0)}\right) \tag{102}$$

or a pair of inequalities

$$\rho\left(\tilde{k}_1,\ 1\right)>\rho\left(k_1^{(0)},\ 1\right)\ \text{ and }\ \delta\left(\tilde{k}_1,\ \tilde{k}_2\right)\geqslant\delta\left(k_1^{(0)},\ k_2^{(0)}\right) \tag{103}$$

holds, whereas for every $\tilde{\mathbf{V}}_1\left(\tilde{k}_1^{(1)},\ \tilde{k}_2^{(1)}\right)\in\tilde{V}$ and $\tilde{\mathbf{V}}_2\left(\tilde{k}_1^{(2)},\ \tilde{k}_2^{(2)}\right)\in\tilde{V}$ either a pair of inequalities

$$\rho\left(\tilde{k}_1^{(1)},\ 1\right)>\rho\left(\tilde{k}_1^{(2)},\ 1\right)\ \text{ and }\ \delta\left(\tilde{k}_1^{(1)},\ \tilde{k}_2^{(1)}\right)<\delta\left(\tilde{k}_1^{(2)},\ \tilde{k}_2^{(2)}\right) \tag{104}$$

or a pair of inequalities

$$\rho\left(\tilde{k}_1^{(1)},\ 1\right)<\rho\left(\tilde{k}_1^{(2)},\ 1\right)\ \text{ and }\ \delta\left(\tilde{k}_1^{(1)},\ \tilde{k}_2^{(1)}\right)>\delta\left(\tilde{k}_1^{(2)},\ \tilde{k}_2^{(2)}\right) \tag{105}$$

is true, or

$$\rho\left(\tilde{k}_1^{(1)},\ 1\right)=\rho\left(\tilde{k}_1^{(2)},\ 1\right)\ \text{ and }\ \delta\left(\tilde{k}_1^{(1)},\ \tilde{k}_2^{(1)}\right)=\delta\left(\tilde{k}_1^{(2)},\ \tilde{k}_2^{(2)}\right). \tag{106}$$

The lists of nodes in subset (101) are denoted by

$$\tilde{K}_1\left(1\right)\subset K_1\left(1\right)\ \text{ and }\ \tilde{K}_2\left(1\right)\subset K_2\left(1\right).$$

Then node $k_1^{***}\in\tilde{K}_1\left(1\right)$ is selected such that the sum

$$\rho\left(k_1^{***},\ 1\right)+\delta\left(k_1^{***},\ k_2^{***}\right) \tag{107}$$

for some vector $\tilde{\mathbf{V}}\left(k_1^{***},\ k_2^{***}\right)\in\tilde{V}$ (where $k_2^{***}\in\tilde{K}_2$) is maximal. Thereupon the second open-loop subtour destination node is the depot, to which the salesman departs from node k_1^{***}.

In general, for $M=2^n$ clusters by (16) — (22) for $n\in\mathbb{N}\backslash\left\{1\right\}$, the starting and destination nodes are determined similarly to (97) — (107). Node $k_{m-1}^{***}\in\tilde{K}_{m-1}\left(n\right)$ is the destination node for cluster $m-1$ and it is the starting node for cluster m, where $m=\overline{2,\ M-1}$. The destination node $k_m^{***}\in\tilde{K}_m\left(n\right)$ for cluster m is selected such that the sum

$$\rho\left(k_m^{***},\ 1\right)+\delta\left(k_m^{***},\ k_{m+1}^{***}\right)$$

is maximal for a vector

$$\begin{gathered}\tilde{\mathbf{V}}\left(k_m^{***},\ k_{m+1}^{***}\right)\in\tilde{V}\subset V=\\ =\left\{\mathbf{V}\left(k_m,\ k_{m+1}\right)=\left[\ \rho\left(k_m,\ 1\right)\quad \delta\left(k_m,\ k_{m+1}\right)\ \right]:\ k_m\in K_m\left(n\right),\ k_{m+1}\in K_{m+1}\left(n\right)\right\},\end{gathered} \tag{108}$$

where $k_{m+1}^{***}\in\tilde{K}_{m+1}\left(n\right)$, and subset $\tilde{V}\subset V$ in (108) by respective denotations

$$\tilde{K}_m\left(n\right)\subset K_m\left(n\right)\ \text{ and }\ \tilde{K}_{m+1}\left(n\right)\subset K_{m+1}\left(n\right)$$

is selected such that for every $\tilde{\mathbf{V}}\left(\tilde{k}_m,\ \tilde{k}_{m+1}\right) \in \tilde{V}$ and every $\mathbf{V}_0\left(k_m^{(0)},\ k_{m+1}^{(0)}\right) \in \left\{V \backslash \tilde{V}\right\}$ either a pair of inequalities

$$\rho\left(\tilde{k}_m,\ 1\right) \geqslant \rho\left(k_m^{(0)},\ 1\right) \ \text{ and } \ \delta\left(\tilde{k}_m,\ \tilde{k}_{m+1}\right) > \delta\left(k_m^{(0)},\ k_{m+1}^{(0)}\right)$$

or a pair of inequalities

$$\rho\left(\tilde{k}_m,\ 1\right) > \rho\left(k_m^{(0)},\ 1\right) \ \text{ and } \ \delta\left(\tilde{k}_m,\ \tilde{k}_{m+1}\right) \geqslant \delta\left(k_m^{(0)},\ k_{m+1}^{(0)}\right)$$

holds, whereas for every $\tilde{\mathbf{V}}_1\left(\tilde{k}_m^{(1)},\ \tilde{k}_{m+1}^{(1)}\right) \in \tilde{V}$ and $\tilde{\mathbf{V}}_2\left(\tilde{k}_m^{(2)},\ \tilde{k}_{m+1}^{(2)}\right) \in \tilde{V}$ either a pair of inequalities

$$\rho\left(\tilde{k}_m^{(1)},\ 1\right) > \rho\left(\tilde{k}_m^{(2)},\ 1\right) \ \text{ and } \ \delta\left(\tilde{k}_m^{(1)},\ \tilde{k}_{m+1}^{(1)}\right) < \delta\left(\tilde{k}_m^{(2)},\ \tilde{k}_{m+1}^{(2)}\right)$$

or a pair of inequalities

$$\rho\left(\tilde{k}_m^{(1)},\ 1\right) < \rho\left(\tilde{k}_m^{(2)},\ 1\right) \ \text{ and } \ \delta\left(\tilde{k}_m^{(1)},\ \tilde{k}_{m+1}^{(1)}\right) > \delta\left(\tilde{k}_m^{(2)},\ \tilde{k}_{m+1}^{(2)}\right)$$

is true, or

$$\rho\left(\tilde{k}_m^{(1)},\ 1\right) = \rho\left(\tilde{k}_m^{(2)},\ 1\right) \ \text{ and } \ \delta\left(\tilde{k}_m^{(1)},\ \tilde{k}_{m+1}^{(1)}\right) = \delta\left(\tilde{k}_m^{(2)},\ \tilde{k}_{m+1}^{(2)}\right).$$

Node $k_{M-1}^{***} \in \tilde{K}_{M-1}(n)$ is the destination node for cluster $M-1$ and it is the starting node for cluster M. The destination node for cluster M is the depot. The open-loop subroutes (subtours) are assembled through the depot and nodes $\{k_m^{***}\}_{m=1}^{M-1}$ making thus a closed-loop route as an approximate solution to the initial TSP. This method of determining connectors $\{k_m^{***}\}_{m=1}^{M-1}$ is especially efficient for a few clusters. It is explained by moving farther away from the depot reduces likelihood of that these nodes are located too close that may lower the quality of an approximate solution. Indeed, if nodes $\{k_m^{***}\}_{m=1}^{M-1}$ are too close, then it is more probable that a route in which they are connected directly one after another is shorter than a route assembled by connecting the M open-loop subroutes through these nodes.

However, the M open-loop subroutes can be connected in a simpler manner, without moving farther away from the depot. For this, the depot is not initially considered as a specific node. The iterative clustering by (16)—(22) is slightly modified. At step n of the clustering, these clusters are still (16) by inclusion

$$K_1(n) \subset K_1(n-1), \tag{109}$$

union-intersection statements (18), and

$$\mathrm{N}_1(1) = \left\{\left[\begin{array}{cc} p_{k_1 1} & p_{k_1 2} \end{array}\right]\right\}_{k_1 \in K_1(1)}, \ \ K_1(1) \subset \left\{\overline{1,\ N}\right\}, \tag{110}$$

$$\mathrm{N}_2(1) = \left\{\left[\begin{array}{cc} p_{k_2 1} & p_{k_2 2} \end{array}\right]\right\}_{k_2 \in K_2(1)}, \ \ K_2(1) \subset \left\{\overline{1,\ N}\right\}, \tag{111}$$

where union-intersection statements (21) and (22) hold. Then

$$\left\{k_m^{**},\ k_{m+1}^{**}\right\} \in \arg \min_{k_m \in K_m(n),\ k_{m+1} \in K_{m+1}(n)} \rho\left(k_m,\ k_{m+1}\right), \ \ k_m^{***} = k_m^{**} \ \text{ for } \ m = \overline{1,\ M-1} \tag{112}$$

and

$$\left\{k_M^{**},\ k_1^{**}\right\} \in \arg \min_{k_M \in K_M(n),\ k_1 \in K_1(n)} \rho\left(k_M,\ k_1\right), \ \ k_M^{***} = k_M^{**}, \tag{113}$$

where k_1^{**} becomes the fictional depot. An example of applying such a technique is shown in Figure 13, where an approximate solution to the well-known Mona Lisa problem [3] is found by the rectangular cell clustering approach for 16 clusters. There are 10^5 nodes and the assembled route length is 6195821.4779, while it is claimed that a lower bound found by the Concorde solver [32, 26] is 5757084, and no shorter route exists. The assembling

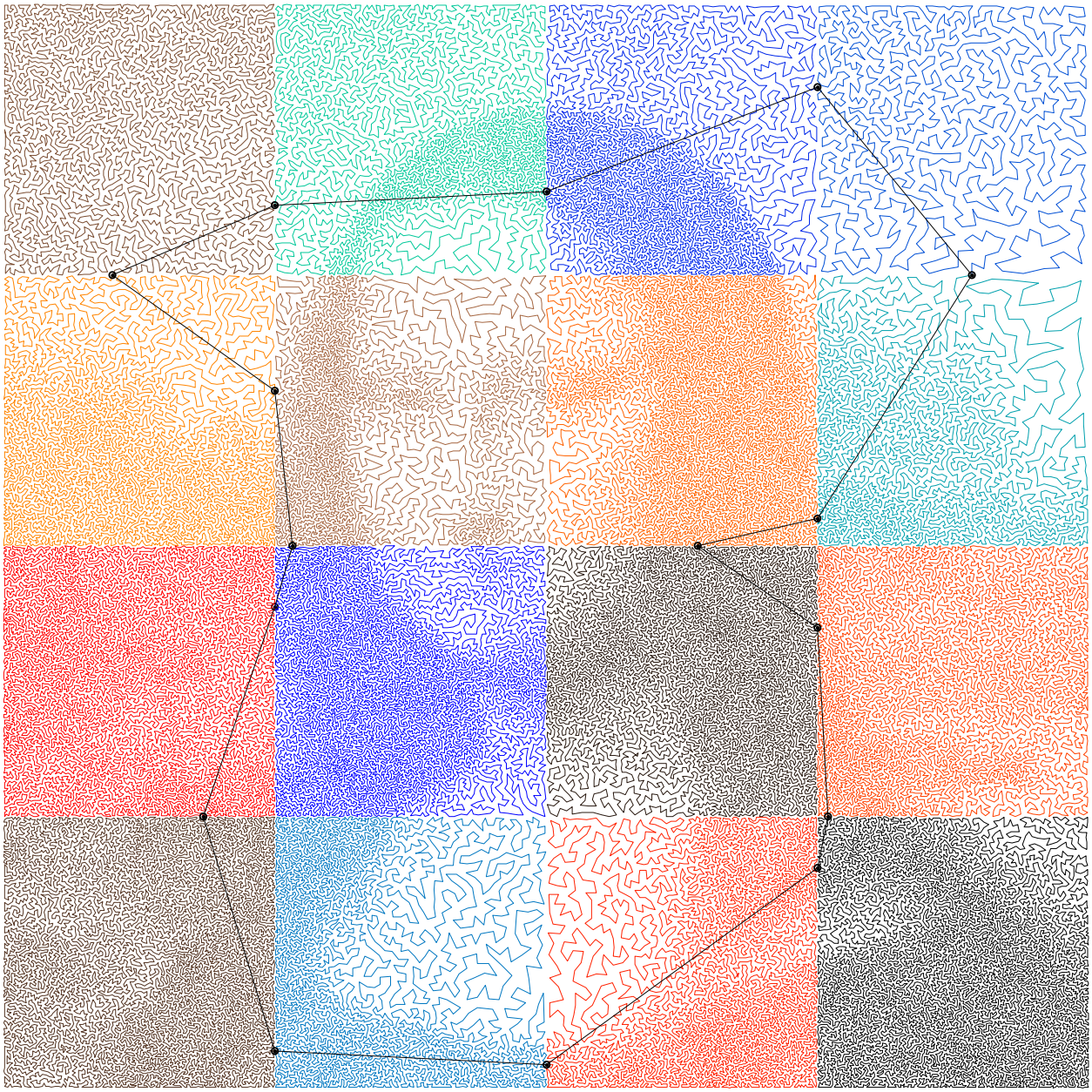

Figure 13. An approximate solution to the Mona Lisa problem obtained with the rectangular cell clustering approach for 16 clusters (highlighted by varying colors)

polyline is also shown along with connectors $\{k_m^{***}\}_{m=1}^{16}$, where the pattern from Figure 6 is used. The computation has lasted for 1597.35 hours on a single CPU Intel Core i5-7200U@2.50GHz. It has taken 148888329 iterations to solve 16 open-loop TSPs whose number of nodes varies between 1124 and 9855. The open-loop TSP with 1124 nodes has taken 427473 iterations, but the open-loop TSP with 9855 nodes taken 17374037 iterations is solved faster than the open-loop TSP with 9815 nodes taken 18498473 iterations.

Nevertheless, the Mona Lisa problem is solved far faster when it is divided into 64 clusters [41, 42]. Figure 14 presents another approximate solution, which is just 0.24 % longer than that in Figure 13. The assembling polyline

is also shown along with connectors $\{k_m^{***}\}_{m=1}^{64}$, where the pattern from Figure 6 is used. The computation has lasted for 186.123 hours on the abovementioned single CPU, which is at least 0.136 % faster than by the existing state-of-the-art parallel approximation algorithms [9, 44, 30, 18]. To solve 64 open-loop TSPs whose number of nodes varies between 158 and 3102, it has taken 53388914 iterations, which is 2.7887 times less than that for the 16 clusters. Thus, the smaller-cluster division ensures a quite significant speedup (it is 8.5822 times faster) by worsening the approximate solution only by 0.24 %. Meanwhile, the resulting Mona Lisa image reconstruction appears to be of much the same quality (Figure 15). In this particular case, the approximate solution in Figure 14 nicely balances the accuracy loss and computational time, making the tradeoff appropriate.

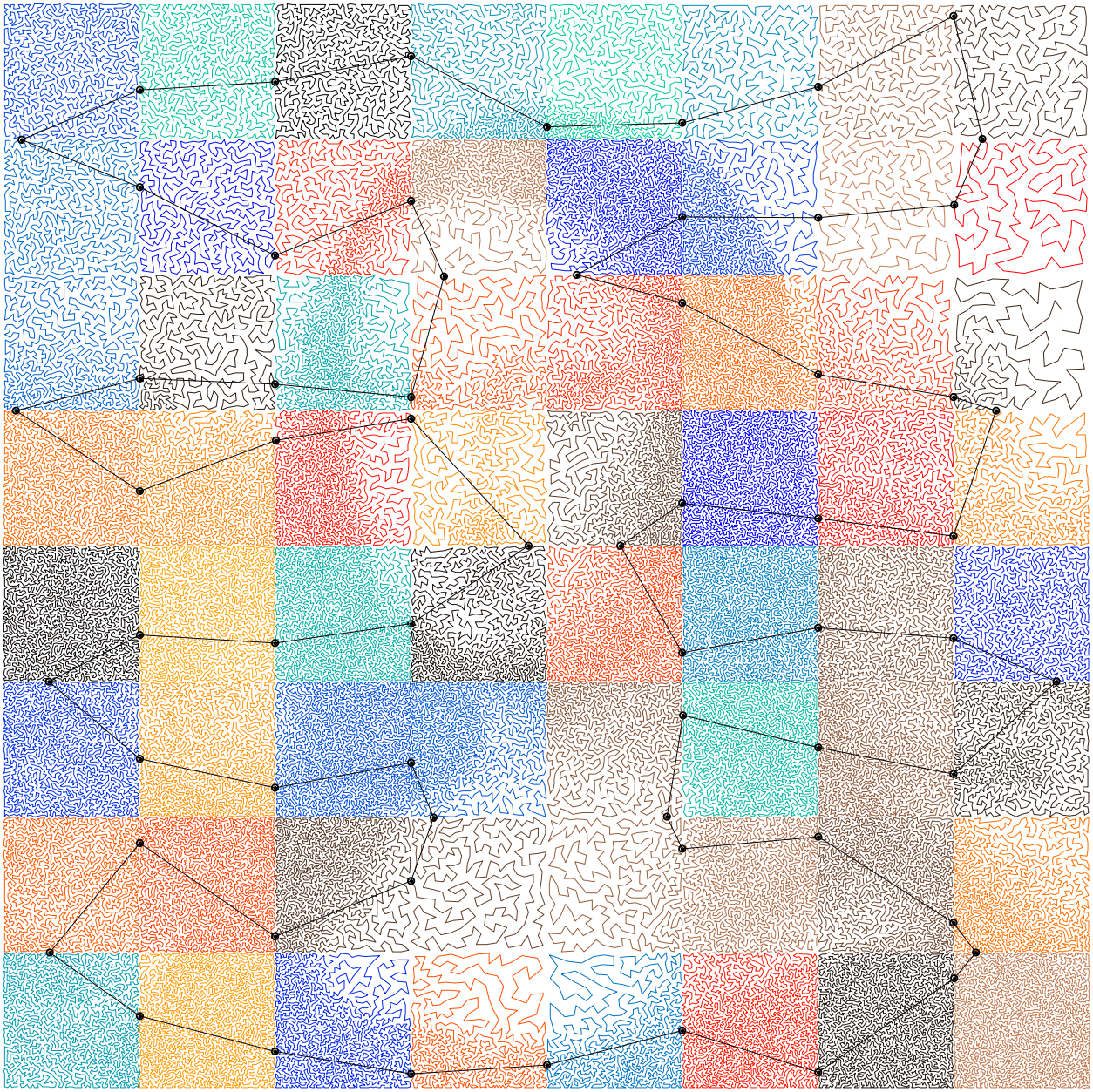

Figure 14. An approximate solution to the Mona Lisa problem obtained with the rectangular cell clustering approach for 64 clusters (highlighted by varying colors); the route is 0.24 % longer than that in Figure 13, but obtained 8.5822 times faster

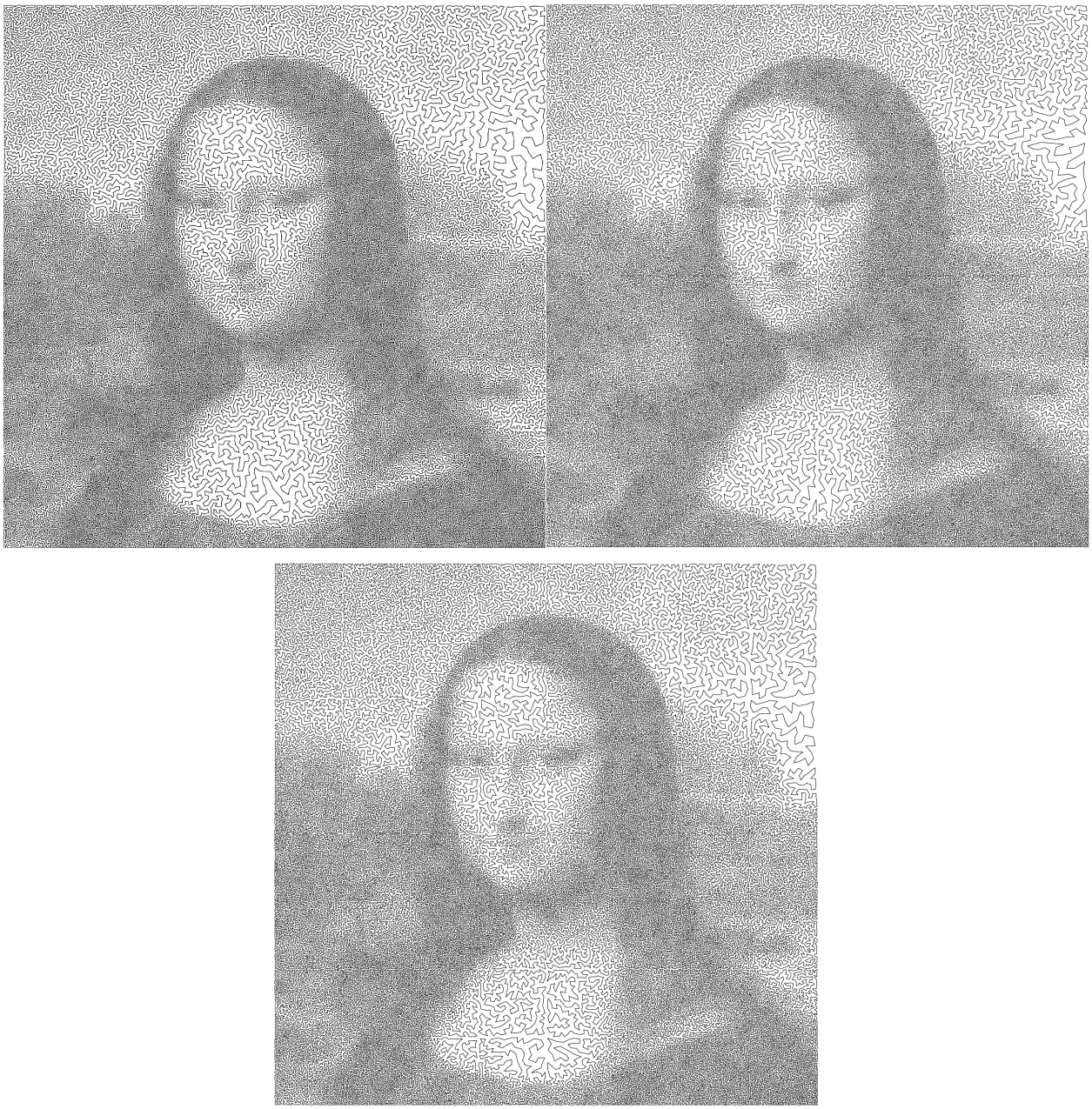

Figure 15. The original image (upper left) as the route drawn by Robert Bosch [3], the 16-clustered route by Figure 13 (upper right), and the 64-clustered route by Figure 14 (bottom)

The tradeoff appropriateness is further strengthened if to consider solving the open-loop TSPs on 16 and 64 parallel processor cores, respectively. If all the open-loop TSPs for Figure 13 are solved on 16 parallel cores, the time spent on solving is the computational time of the open-loop TSP with 9815 nodes, which is 247.2985 hours. If all the open-loop TSPs for Figure 14 are solved on 64 parallel cores, the computation lasts at most for 10 hours 39 minutes (it is the computational time of the open-loop TSP with 3102 nodes). Hence, the 64-clustered route still being only 0.24 % longer is obtained 23.2205 times faster than the 16-clustered route by solving open-loop TSPs in parallel.

9. Discussion of efficiency and limitations

It is apparent that more squarish datasets fit better the rectangular cell clustering pattern. As both the rectangular assembling and the centroid TSP assembling approaches are slower, the rectangular cell clustering pattern is way more efficient than the iterative clustering on squarish datasets. Meanwhile, the accuracies of these three approaches are comparable, even when the dataset becomes less squarish and more roundish, oval, ellipsoidal, etc.

For a given TSP, the rectangular cell clustering pattern can be used for fast approximation of the TSP shortest route. An approximately shortest route suggests two things. First, it serves as an upper bound. Due to the uncertainty of the approximate solution, which depends on the pseudorandom number generator initial state [36, 52, 6, 39, 28, 23], the upper bound may be lowered by re-solving the TSP for a few times in a row or in parallel. Second, the route can be studied and scrutinized to find its bottlenecks (or vulnerable zones), which subsequently are separated and another bunch of open-loop TSPs is approximately solved.

Another merit of the rectangular cell clustering pattern is the parallelization efficiency. The pattern quickly makes solving any TSP parallelizable. It is true for big-sized TSPs similar to that in Figures 12 — 14. Availability of multitudinous parallel processor cores is not a trouble today. Besides, the processors are not really required to be perfectly synchronized, unlike other parallel-based approaches [9, 18, 44, 30, 53]. For that matter, a TSP with a million nodes similar to the Mona Lisa problem can be approximately solved within an hour by using 1024 processor cores (here the average number of nodes per cluster is 976.5625, which requires about 45 minutes). A TSP with a billion nodes, to be approximately solved within an hour or so, would require 1048576 processor cores. If this amount is not available, the TSP is parallelized on an available amount. For instance, if 1024 processor cores are available, then the respective 1024×1024 rectangular lattice is imposed on a billion nodes, and 1024 bunches of 1024 open-loop TSPs are solved within roughly 1024 hours, which is 43 days.

The key limitation of the suggested approach is its dependence upon squarish datasets. However, if a dataset is of an irregular nonconvex shape, its protuberances can be cut off as separate open-loop TSPs to obtain and solve an open-loop "mother" TSP (of roughly a convex shape). Thereupon the approximate solutions of the open-loop TSPs, including the "mother" TSP are assembled. Such an approach fits serpentine-like datasets as well [19, 43].

The assemblage mostly implying a selection of the connectors is also important. The suggested assemblage in accordance with (62) — (67) exemplified in Figures 6 and 7 is not the only possible one. There are closed-loop serpentines of other forms. For instance, the serpentine for a 4×4 rectangular lattice reminds the letter I (see the second row right subplot in Figure 6), but it can be "transposed" so that it will remind the letter H.

Another limitation is the 2^n pattern. For instance, if a dataset is still squarish, but its shape has an aspect ratio of 3-to-1, applying a 2×4 lattice (or 4×8, 8×16, 16×32, etc.) will result in stretched clusters. This may lead to accuracy losses from inefficient horizontal connections in the assemblage. However, a more relevant, not necessarily 2^n-pattern, rectangular lattice can be applied by subsequently mixing it with the centroid TSP assembling approach. Therein, the iterative clustering must be modified so that not all clusters are further divided at a step n of the clustering, but only those whose size is not sufficiently small. In this case, the iterative clustering resembles a binary tree with pruned branches, whose leaves are clusters of a sufficiently small size. The criteria for deciding when open-loop TSPs are sufficiently small depend on the topology of nodes, though. Overall, the sufficient smallness cannot be certainly formalized.

If density of nodes badly varies, it does not seem to affect the accuracy heavily. Indeed, the Mona Lisa problem has visible parts of badly varying density (hair and shoulders against the upper part of the background and the other light parts like face and neck), but the two divisions into 16 and 64 clusters (Figures 12 and 13) are not followed by a significant accuracy difference. Therefore, it is expected that the suggested approach can successfully handle real-world TSPs with capacity limits, time windows, vehicle speed inconstancy, etc., because such additional constraints are embedded into the genetic algorithm [49, 16, 41, 42].

The comparative analysis of Tables 1 — 5 has shown that the accuracy gain and speedup, if any, is not stable through fewer TSPs. Nevertheless, Table 1 confirms the parallelization gain by using the rectangular cell clustering pattern is positively scalable — as either the TSP size increases or the number of clusters is reasonably increased, or they both increase, the gain grows.

10. Conclusion

This paper basically suggests two ways of dividing a closed-loop TSP into smaller-sized open-loop TSPs: the rectangular cell clustering and iterative clustering. The latter has two ways to assemble the solutions of the open-loop TSPs: the rectangular assembling approach, by which the assemblage is done via a symmetric rectangular closed-loop serpentine, and the centroid TSP assembling approach, which requires solving a supplementary closed-loop TSP whose nodes are centroids of the open-loop-TSP clusters. The main intention of the clustering is to parallelize the TSP for quickly determining its approximate solution that serves as an upper bound of the TSP solution whose bottlenecks can be studied also in the approximation.

While the rectangular cell clustering is determined by only the lattice size, the iterative clustering is primarily determined by the clustering method. The method for clustering can be any method allowing to efficiently divide a set of nodes into two groups by minimizing the distance within the group and maximizing the distance between the groups, taking into consideration possible variation of node density.

Based on the results obtained from the numerical simulation, it is ascertained that both the rectangular cell clustering pattern and iterative clustering are roughly peers at the accuracy, but a significant difference exists in the computational time. An approximate solution to the TSP is obtained faster by the rectangular cell clustering pattern, whereas it performs much better on squarish datasets. The main scientific contribution consists in further improving the approaches to an efficient approximate analysis of closed-loop TSPs by shortening the computational time and not exceeding tolerable accuracy losses being uncertain for big-sized TSPs. This is done by using a subset of nodes connecting open-loop TSPs via either a rectangular closed-loop serpentine or the polyline from a solution of the supplementary centroid TSP. The suggested approach has a significant impact and practical contribution as it allows approximately solving big-sized TSPs on multitudinous parallel processor cores without requiring their synchronization.

The research can be extended onto building approximately shortest routes accomplished by multiple salesmen. Although such TSPs are still parallelizable, their feasible solutions must obey specific constraints issuing from the multiplicity of salesmen. In addition, minimization of the number of salesmen may be an additional criterion of solution optimality.

Acknowledgement

The work was technically supported by the Faculty of Mechanical and Electrical Engineering at the Polish Naval Academy, Gdynia, Poland.

REFERENCES

1. D. L. Applegate, R. Bixby, V. Chvátal, and W. J. Cook, *The Traveling Salesman Problem: A Computational Study*, Princeton University Press, Princeton, NJ, USA, 2007.
2. C. Archetti, L. Peirano, and M. G. Speranza, *Optimization in multimodal freight transportation problems: A Survey*, European Journal of Operational Research, vol. 299, iss. 1, pp. 1–20, 2022.
3. R. Bosch and A. Herman, *Continuous line drawings via the traveling salesman problem*, Operations Research Letters, vol. 32, iss. 4, pp. 302–303, 2004.
4. L. D. Chambers, *The Practical Handbook of Genetic Algorithms*, Chapman and Hall/CRC, Boca Raton, FL, USA, 2000.
5. O. Cheikhrouhou and I. Khoufi, *A comprehensive survey on the Multiple Traveling Salesman Problem: Applications, approaches and taxonomy*, Computer Science Review, vol. 40, Article no. 100369, 2021.
6. A. Colorni, M. Dorigo, F. Maffioli, V. Maniezzo, G. Righini, and M. Trubian, *Heuristics from nature for hard combinatorial optimization problems*, International Transactions in Operational Research, vol. 3, iss. 1, pp. 1–21, 1996.
7. C. Ding, Y. Cheng, and M. He, *Two-level genetic algorithm for clustered traveling salesman problem with application in large-scale TSPs*, Tsinghua Science & Technology, vol. 12, iss. 4, pp. 459–465, 2007.
8. S. Dutta, *Approximate spatial reasoning: integrating qualitative and quantitative constraints*, International Journal of Approximate Reasoning, vol. 5, iss. 3, pp. 307–331, 1991.
9. C. N. Fiechter, *A parallel tabu search algorithm for large traveling salesman problems*, Discrete Applied Mathematics, vol. 51, pp. 243–267, 1994.
10. M. Fischetti, A. Lodi, and P. Toth, *Exact methods for the asymmetric traveling salesman problem*, in: G. Gutin and A. P. Punnen (Eds.), *The Traveling Salesman Problem and Its Variations*, Kluwer, Boston, MA, USA, 2002, pp. 169–205.

11. L.-A. Gottlieb, R. Krauthgamer, and H. Rika, *Faster algorithms for orienteering and k-TSP*, Theoretical Computer Science, vol. 914, pp. 73–83, 2022.
12. J. C. Gower and G. J. S. Ross, *Minimum spanning trees and single linkage cluster analysis*, Applied Statistics, vol. 18, iss. 1, pp. 54–64, 1969.
13. A. P. Guerreiro, J. Cortes, D. Vanderpooten, C. Bazgan, I. Lynce, V. Manquinho, and J. R. Figueira, *Exact and approximate determination of the Pareto front using Minimal Correction Subsets*, Computers & Operations Research, vol. 153, Article no. 106153, 2023.
14. J. A. Hartigan and M. A. Wong, *Algorithm AS 136: A k-means clustering algorithm*, Journal of the Royal Statistical Society, Series C, vol. 28, iss. 1, pp. 100–108, 1979.
15. R. L. Haupt and S. E. Haupt, *Practical Genetic Algorithms*, John Wiley & Sons, Hoboken, NJ, USA, 2003.
16. N. Helal, F. Pichon, D. Porumbel, D. Mercier, and É. Lefèvre, *The capacitated vehicle routing problem with evidential demands*, International Journal of Approximate Reasoning, vol. 95, pp. 124–151, 2018.
17. A. Hertz and M. Widmer, *Guidelines for the use of meta-heuristics in combinatorial optimization*, European Journal of Operational Research, vol. 151, iss. 2, pp. 247–252, 2003.
18. K. Honda, Y. Nagata, and I. Ono, *A parallel genetic algorithm with edge assembly crossover for 100,000-city scale TSPs*, Proceedings of the 2013 IEEE Congress on Evolutionary Computation, pp. 1278–1285, 2013.
19. R. Inoue, S. Shiode, and N. Shiode, *Detection of irregular-shaped clusters on a network by controlling the shape compactness with a penalty function*, GeoJournal, vol. 88, pp. 3817–3832, 2023.
20. A. Király and J. Abonyi, *Redesign of the supply of mobile mechanics based on a novel genetic optimization algorithm using Google Maps API*, Engineering Applications of Artificial Intelligence, vol. 38, pp. 122–130, 2015.
21. L. Kota and K. Jarmai, *Mathematical modeling of multiple tour multiple traveling salesman problem using evolutionary programming*, Applied Mathematical Modelling, vol. 39, iss. 12, pp. 3410–3433, 2015.
22. A. Land, *The solution of some 100-city travelling salesman problems*, EURO Journal on Computational Optimization, vol. 9, Article no. 100017, 2021.
23. G. Laporte, *The traveling salesman problem: An overview of exact and approximate algorithms*, European Journal of Operational Research, vol. 59, iss. 2, pp. 231–247, 1992.
24. A. LaTorre, J. M. Peña, V. Robles, and S. Muelas, *Using multiple offspring sampling to guide genetic algorithms to solve permutation problems*, in: Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation, ACM, New York, NY, USA, 2008, pp. 1119–1120.
25. E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, and D. B. Shmoys, *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization*, Wiley, Chichester, UK, 1985.
26. T. Lechien, J. Jooken, and P. De Causmaecker, *Evolving test instances of the Hamiltonian completion problem*, Computers & Operations Research, vol. 149, Article no. 106019, 2023.
27. C. Leopold, *Arranging program statements for locality on the basis of neighbourhood preferences*, International Journal of Approximate Reasoning, vol. 19, iss. 1–2, pp. 73–90, 1998.
28. J. Li and M. Li, *An analysis on convergence and convergence rate estimate of elitist genetic algorithms in noisy environments*, Optik, vol. 124, iss. 24, pp. 6780–6785, 2013.
29. J. N. Macgregor and T. Ormerod, *Human performance on the traveling salesman problem*, Perception & Psychophysics, vol. 58, iss. 4, pp. 527–539, 1996.
30. M. Manfrin, M. Birattari, T. Stützle, and M. Dorigo, *Parallel ant colony optimization for the traveling salesman problem*, in: M. Dorigo, L. M. Gambardella, M. Birattari, A. Martinoli, R. Poli, and T. Stützle (Eds.), *Ant Colony Optimization and Swarm Intelligence. ANTS 2006*, in: Lecture Notes in Computer Science, vol. 4150, 2006, pp. 224–234.
31. A. Maya-López, A. Martínez-Ballesté, and F. Casino, *A compression strategy for an efficient TSP-based microaggregation*, Expert Systems with Applications, vol. 213, Part B, Article no. 118980, 2023.
32. K. Michalak, *Feasibility-preserving genetic operators for hybrid algorithms using TSP solvers for the Inventory Routing Problem*, Procedia Computer Science, vol. 192, pp. 1451–1460, 2021.
33. S. A. Mulder and D. C. Wunsch, *Million city traveling salesman problem solution by divide and conquer clustering with adaptive resonance neural networks*, Neural Networks, vol. 16, iss. 5, pp. 827–832, 2003.
34. A. J. Orman and H. P. Williams, *A Survey of Different Integer Programming Formulations of the Travelling Salesman Problem*, in: E. Kontoghiorghes and C. Gatu (Eds.), *Optimisation, Econometric and Financial Analysis. Advances in Computational Management Science*, vol. 9, Springer, Berlin, Heidelberg, Germany, 2006, pp. 91–104.
35. R. Pan, Z. Zhang, Y. Fan, J. Cao, K. Lu, and T. Yang, *Multi-objective optimization method for learning thresholds in a decision-theoretic rough set model*, International Journal of Approximate Reasoning, vol. 71, pp. 34–49, 2016.
36. C. Rego, D. Gamboa, F. Glover, and C. Osterman, *Traveling salesman problem heuristics: Leading methods, implementations and latest advances*, European Journal of Operational Research, vol. 211, iss. 3, pp. 427–441, 2011.
37. J. A. Rojas Cruz and A. G. C. Pereira, *The elitist non-homogeneous genetic algorithm: Almost sure convergence*, Statistics & Probability Letters, vol. 83, iss. 10, pp. 2179–2185, 2013.
38. V. V. Romanuke, A. Y. Romanov, and M. O. Malaksiano, *Crossover operators in a genetic algorithm for maritime cargo delivery optimization*, Journal of ETA Maritime Science, vol. 10, iss. 4, pp. 223–236, 2022.
39. V. V. Romanuke, A. Y. Romanov, and M. O. Malaksiano, *Pseudorandom number generator influence on the genetic algorithm performance to minimize maritime cargo delivery route length*, Scientific Journal of Maritime Research, vol. 36, pp. 249–262, 2022.
40. V. V. Romanuke, *Speedup of the k-means algorithm for partitioning large datasets of flat points by a preliminary partition and selecting initial centroids*, Applied Computer Systems, vol. 28, no. 1, pp. 1–12, 2023.
41. V. V. Romanuke, *Traveling salesman problem parallelization by solving clustered subproblems*, Foundations of Computing and Decision Sciences, vol. 48, no. 4, pp. 453–481, 2023.
42. V. V. Romanuke, *Deep clustering of the traveling salesman problem to parallelize its solution*, Computers & Operations Research, vol. 165, Article no. 106548, 2024.

43. B. Schelling and C. Plant, *Dataset-Transformation: improving clustering by enhancing the structure with DipScaling and DipTransformation*, Knowledge and Information Systems, vol. 62, pp. 457–484, 2020.
44. J. Schneider, C. Froschhammer, I. Morgenstern, T. Husslein, and J. M. Singer, *Searching for backbones — an efficient parallel algorithm for the traveling salesman problem*, Computer Physics Communications, vol. 96, iss. 2–3, pp. 173–188, 1996.
45. C. A. Silva, J. M. C. Sousa, T. Runkler, and R. Palm, *Soft computing optimization methods applied to logistic processes*, International Journal of Approximate Reasoning, vol. 40, iss. 3, pp. 280–301, 2005.
46. C. Song, K. Lee, and W. D. Lee, *Extended simulated annealing for augmented TSP and multi-salesmen TSP*, in: Proceedings of the International Joint Conference on Neural Networks, Portland, OR, USA, 2003, pp. 2340–2343.
47. X. M. Song, B. Li, and H. M. Yang, *Improved Ant Colony Algorithm and its Applications in TSP*, in: Proceedings of Intelligent Systems Design and Applications, Jian, China, 2006, pp. 1145–1148.
48. B. Toaza and D. Esztergár-Kiss, *A review of metaheuristic algorithms for solving TSP-based scheduling optimization problems*, Applied Soft Computing, vol. 148, Article no. 110908, 2023.
49. P. Toth and D. Vigo, *Branch-and-bound algorithms for the capacitated VRP*, in: P. Toth and D. Vigo (Eds.), *The vehicle routing problem*, SIAM Monographs on Discrete Mathematics and Applications, SIAM, Philadelphia, PA, USA, 2002, pp. 29–51.
50. C. L. Valenzuela and A. J. Jones, *Evolutionary divide and conquer (I): A novel genetic approach to the TSP*, Evolutionary Computation, vol. 1, iss. 4, pp. 313–333, 1993.
51. B. van Stein, H. Wang, W. Kowalczyk, M. Emmerich, and T. Bäck, *Cluster-based Kriging approximation algorithms for complexity reduction*, Applied Intelligence, vol. 50, iss. 3, pp. 778–791, 2020.
52. P. Zhang, J. Wang, Z. Tian, S. Sun, J. Li, and J. Yang, *A genetic algorithm with jumping gene and heuristic operators for traveling salesman problem*, Applied Soft Computing, vol. 127, Article no. 109339, 2022.
53. Y. Zhou, F. He, N. Hou, and Y. Qiu, *Parallel ant colony optimization on multi-core SIMD CPUs*, Future Generation Computer Systems, vol. 79, Part 2, pp. 473–487, 2018.