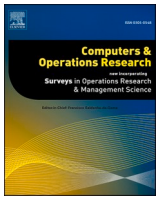
Computers & Operations Research 165 (2024) 106548

Contents lists available at ScienceDirect

Computers and Operations Research

journal homepage: www.elsevier.com/locate/cor



Deep clustering of the traveling salesman problem to parallelize its solution

Vadim V. Romanuke *

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

ARTICLE INFO

Keywords:
Traveling salesman problem
Clustered subproblems
Deep clustering
Parallelization
Route length
Genetic algorithm

ABSTRACT

A method of heuristically solving large traveling salesman problems is suggested, where a dramatic computational speedup is guaranteed. A specific genetic algorithm is the solver. The initial problem is broken into a set of open-loop subproblems by clustering the nodes. First, the nodes are broken into just two clusters. If these open-loop subproblems are tractable and can be solved within reasonable amount of time, no further clustering is needed. Otherwise, the clustering pattern is applied once again, and each of the previous clusters is broken into just two clusters. Thus, the clusters evolve being re-clustered deeper until the size of the respective open-loop subproblems becomes sufficiently small. Therefore, the number of clusters is a power of 2 depending on the number of available processors and available memory with the maximum possible array. The cluster open-loop subproblems are quickly solved in any order and then their solutions as open-loop subroutes are aggregated into the close-loop route as an approximate solution of the initial problem. By the centroid-based approach, the open-loop subroutes are aggregated by the shortest closed loop route passing through the centroids of the clusters. The aggregation route is found as the respective supplementary traveling salesman problem solution. By the serpentine-based approach, the open-loop subroutes are aggregated by a symmetric rectangular close-loop serpentine passing through the rectangular lattice cell centers approximately corresponding to the centroids.

1. Introduction to deep clustering

It is well experienced that in solving a smaller-sized traveling salesman problem (TSP) the computation period of an iteration is shorter than the computation period of an iteration in solving a bigger-sized TSP. This is a concern as of exact methods, as well as of heuristic methods (Fischetti et al., 2002; Hertz and Widmer, 2003). Finding the exact solution for the TSP being an NP-hard problem in combinatorial optimization (Lawler et al., 1985; Du and Pardalos, 1998) becomes roughly exponentially intractable as the number of nodes (sometimes referred to as cities, ports, links, etc.) is increased starting off a few tens to a few hundreds (Fischetti et al., 2002; Applegate et al., 2007; Woeginger, 2003). The intractability by the menioned TSP size frequently arises in time-dependent TSPs (Balma et al., 2023), as well as in TSPs with node priorities (Teixeira and de Araujo, 2024). Various branch-and-bound algorithms and progressive improvement algorithms using techniques reminiscent of linear programming can solve TSPs within reasonable amount of time by particular conditions with up to 200 nodes but no larger than that (Lawler et al., 1985; Toth and Vigo, 2002; Land, 2021). Compared to the approaches, which aim at finding the exact solution, heuristic algorithms perform extremely fast by rapidly converging and approximating to exact solutions (Rego et al., 2011; Colorni et al., 1996). This saves operational time and computational resources (Chambers, 2000; Kota and Jarmai, 2015) in roughly exponential amounts as the number of nodes is increased. Meanwhile, the convergence rate depends on evolution strategies used (Hansen and Ostermeier, 2001; Kramer, 2010; Wang et al., 2014). Although it is unknown how close the exact (optimal) and approximate solutions are, it is highly probable that the latter are at most 3 % away from the optimal solution (Hertz and Widmer, 2003; Du and Pardalos, 1998; Archetti et al., 2022; Haupt and Haupt, 2003). This is also true for extremely large TSPs with millions of nodes (Mulder and Wunsch, 2003).

Designed for combinatorial optimization in general, genetic algorithms are very fast in solving the TSP (Chambers, 2000; Haupt and Haupt, 2003; LaTorre et al., 2008; Zhang et al., 2022; Bäck et al., 2023; Dasgupta and Michalewicz, 2013). Usually they are faster than the algorithms of ant colony optimization (Song et al., 2006), simulated annealing (Song et al., 2003), particle swarm optimization (Zhong et al., 2018), and tabu search (Fiechter, 1994). The genetic algorithm domination has been recently shown by Toaza and Esztergár-Kiss (2023). However, due to the genetic algorithm iteration is run too slow in

* Address: 69 Śmidowicza Street, Gdynia 81-127, Poland.
E-mail address: v.romanuke@amw.gdynia.pl.

https://doi.org/10.1016/j.cor.2024.106548
Received 16 August 2023; Received in revised form 10 January 2024; Accepted 11 January 2024
Available online 18 January 2024

solving problems with a few tens of thousands of nodes or so, even the approximate solution cannot be found within reasonable amount of time. Such an issue can be solved by breaking the initial TSP into smaller ones and aggregating their solutions. The smaller subproblems are open-loop TSPs, where the starting and destination nodes are different. This differs from studying multiple solutions of the same TSP in searching for its backbones (Schneider et al., 1996). Parallel ant colony optimization algorithms also rely on solving a TSP by several colonies connected according to a topology (Manfrin et al., 2006; Song et al., 2006), rather than dividing the TSP into smaller TSPs. On the other hand, 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 (Valenzuela and Jones, 1993). This also resembles the parallel tabu search algorithm (Fiechter, 1994), where several moves are performed concurrently.

The breaking can be fulfilled by just clustering the nodes — see, e. g., Mulder and Wunsch (2003), Van Stein et al. (2020). The clustering does not apply a priority rule that would capture the tradeoff between total distance and node priorities (Teixeira and de Araujo, 2024) because commonly nodes in medium-sized TSPs and larger are of equal priorities. By the simplest approach, the nodes are broken into just two clusters (subproblems). If these open-loop TSPs are tractable and can be solved within reasonable amount of time, no further clustering is needed. Otherwise, the clustering pattern is applied once again — each of the previous clusters is broken into just two clusters. Thus, the clusters evolve: they are re-clustered deeper until the size of the respective open-loop TSPs becomes sufficiently small.

2. Motivation and goal

Definite mutation operations in the genetic algorithm can be modified so that the algorithm iteration will run faster owing to making fewer computations (Shafiee et al., 2016; Davendra and Zelinka, 2010). This speeds up the chromosome selection, which is the engine of the genetic algorithm (Haupt and Haupt, 2003; Zheng et al., 2023). Nevertheless, the speedup is tiny, and the algorithm converges significantly slower as the number of nodes increases (Li and Li, 2013; Rojas Cruz and Pereira, 2013).

Theoretically, a significant speedup is possible only when a big-sized TSP is divided into smaller-sized subproblems whose convergence rate is far better than that of the big-sized TSP. The division should be such that the open-loop subroutes must not share nodes except for the nodes at which the subroutes are aggregated into the close-loop route as an approximate solution of the big-sized TSP.

Clustering the nodes of a big-sized TSP is one of the easiest-to-see decisions to implement the division (Celebi et al., 2013; Ikotun et al., 2023). 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 subroute ends. This node serves as a starting node for the subsequent subproblem by whose solution the salesman completes the TSP route. If two clusters are insufficient by some reason, the clustering must go deeper to solve the smallest possible subproblems. It is arguable of how to select nodes at which the solutions of the subproblems are aggregated (Mulder and Wunsch, 2003).

Formally speaking, the TSP parallelization makes sense if the maximal computational time spent on solving the open-loop TSPs in parallel is less than that spent on solving the initial TSP, by not much worsening the accuracy. Otherwise, the initial TSP (which is coined as the whole problem or whole TSP for further consideration) is solved faster without its parallelization. However, it is not always possible to solve simultaneously all the open-loop TSPs in parallel. Moreover, the case when it is only possible to solve one subproblem on a processor cannot be excluded. This implies that, in this very case, the TSP parallelization makes sense if the computational time spent on successively solving all the open-loop TSPs is less than that spent on solving the whole TSP, by not much worsening the accuracy. In addition, the time spent on creating the subproblems and aggregating their solutions must be taken into consideration. Therefore, the primary goal is to ascertain whether the parallelization by the deep clustering is efficient. The efficiency criterion is to shorten the computational time spent on solving all the open-loop TSPs by not lengthening much the aggregated route. If the parallelization is efficient, the secondary goal is to approximately determine limits within which the efficiency is predictable and likely to be maintained. For achieving the goal, the following seven tasks are to be fulfilled:

1. To describe TSP variables, flags and constraints.
2. To formulate the TSP objective to be minimized.
3. To suggest a method by which the initial TSP is quickly broken into open-loop subproblems and the initial TSP solution is efficiently aggregated.
4. To describe basic steps in the genetic algorithm and conditions by which the subproblems can be formulated and solved independently or in parallel.
5. To obtain performance statistics of the parallelization versus solving the whole TSP. The statistics must include the computational time and route length. The initial TSP breaking and the subproblems solutions aggregation performance is to be considered as well.
6. To discuss the parallelization efficiency. To mention the practical applicability, significance, and limitations of the suggested parallelization approach. To estimate an expected gain by embedding the parallelization in genetic algorithms for solving large (a few hundred thousands of nodes) and extremely large problems (millions of nodes).
7. Based on the results obtained and discussed, to conclude on the contribution to the field of genetic algorithms and the TSP.

The paper is structured based on those tasks. Section 3 describe TSP variables, flags and constraints, whereupon Section 4 formulates the TSP objective which is to be minimized. Section 5 presents a quick clustering method, which is alternatively referred to as the deep clustering, for breaking the initial TSP into smaller open-loop TSPs. Section 6 describes basic steps in the genetic algorithm and conditions by which the open-loop TSPs can be formulated and solved independently. Section 7 presents computer simulation statistics of performance of the suggested parallelization approach, where an expected gain is estimated. The results are discussed in Section 8, where solving large-scale TSPs is additionally shown. Finally, Section 9 presents concluding remarks and a further research focus.

3. TSP variables, flags and constraints

In the TSP of N nodes, let the depot (a node of the salesman's starting departure and the ending arrival) have number 1. The TSP is flat, i. e. the salesman's tour has only two-coordinate node locations, without ascensions or descents (Lawler et al., 1985; Toth and Vigo, 2002; Crowder and Padberg, 1980; Grötschel and Holland, 1991; Tayarani-N and Prügel-Bennett, 2016): the set of nodes is a map of two-coordinate node locations with horizontal p_{k1} and vertical p_{k2} components of the location of node k, $k = \overline{1,\ N}$.

If a particular tour from node k to node j is accomplished directly, without visiting intermediate (any of other $N-2$) nodes, then it is done by a straight line. The distance

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

between nodes k and j is covered by the salesman with some constant speed. Therefore, distances (1) can be mapped into time or other units implying the cost of completing the TSP route.

Flags in the TSP show which nodes are connected. The flags are binary (Orman and Williams, 2006):

$x_{kj} \in \{0,\ 1\}$ by $k = \overline{1,\ N}$ and $j = \overline{1,\ N}$. (2)

Obviously, $x_{kk} = 0\ \forall k\ = \overline{1,\ N}$. If the salesman visits node j directly after node k, then this is flagged as $x_{kj}\ = 1$; otherwise $x_{kj}\ = 0$. Moreover, the direct connection of nodes k and j implies only one nonzero flag:

$x_{jk} = 0$ if $x_{kj} = 1$ and $x_{kj} = 0$ if $x_{jk} = 1$. (3)

In this way, surplus flagging is avoided.

There is only one departure from node k towards only one following node:

$$\sum_{j=1}^{N} x_{kj} = 1\ \forall k = \overline{1,\ N}. \tag{4}$$

Symmetrically, there is only one arrival at node j from only one node:

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

The depart-and-arrive logic of the TSP is supplemented by the third constraint. Any subtour

$Q \subset \{\overline{1,\ N}\}$ by $2 \leqslant |Q| < N$ (6)

of the salesman is eliminated by imposing the following requirement:

$$\sum_{k\in Q}\sum_{j\in Q} x_{kj} \leqslant |Q| - 1. \tag{7}$$

Inequality (7) by (6) ensures that no subset Q can form a subtour as a closed loop, and so the route is a single tour being not a union of smaller tours (Dantzig, 1963; Papadimitriou and Steiglitz, 1998).

4. Objective to be minimized

In the TSP of N nodes, there are $0.5N(N-1)$ nonzero symmetric distances

$$\left\{\{\rho(k,\ j)\}_{k=1}^{N-1}\right\}_{j=k+1}^{N} \tag{8}$$

calculated by (1). The route is covered by a part of distances (8). The respective objective function

$$\rho_{\Sigma}\left(\left\{\left\{x_{kj}\right\}_{k=1}^{N}\right\}_{j=1}^{N}\right) = \sum_{k=1}^{N}\sum_{j=1}^{N} x_{kj}\cdot\rho(k,\ j) \tag{9}$$

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

$$x_{kj}^{*} \in \{0,\ 1\} \text{ for } \ k = \overline{1,\ N} \text{ and } j = \overline{1,\ N} \tag{10}$$

at which

$$\begin{aligned}\sum_{k=1}^{N}\sum_{j=1}^{N} x_{kj}^{*}\cdot\rho(k,\ j) &= \rho_{\Sigma}\left(\left\{\left\{x_{kj}^{*}\right\}_{k=1}^{N}\right\}_{j=1}^{N}\right)\\ = \rho_{\Sigma}^{*} &= \min_{\left\{\left\{x_{kj}\right\}_{k=1}^{N}\right\}_{j=1}^{N}} \sum_{k=1}^{N}\sum_{j=1}^{N} x_{kj}\cdot\rho(k,\ j).\end{aligned} \tag{11}$$

The TSP solution given formally as

$$\left\{\left\{x_{kj}^{*}\right\}_{k=1}^{N}\right\}_{j=1}^{N} \tag{12}$$

allows to build the most rational (called optimal) route whose length is the shortest. The TSP may have multiple solutions (12) having the same length ρ_{Σ}^{*}.

5. Deep clustering

The quickest breaking of a TSP into open-loop subproblems is to break the set of initial nodes into just two clusters. This is so because the clustering is performed slower as the number of clusters is increased (Hartigan and Wong, 1979; Celebi et al., 2013; Ikotun et al., 2023). Therefore, dividing a set into two clusters is performed the quickest. Thus, a set of node locations

$$\mathrm{N} = \{[p_{k1}\quad p_{k2}]\}_{k=1}^{N} \tag{13}$$

is divided into just two groups: a first group

$$\mathrm{N}_1(1) = \left\{\left[p_{k_1 1}\quad p_{k_1 2}\right]\right\}_{k_1\in K_1(1)} \tag{14}$$

of nodes by a set of node indices $K_1(1) \subset \{\overline{1,\ N}\}$ in the first group, $1 \in K_1(1)$, and a second group

$$\mathrm{N}_2(1) = \left\{\left[p_{k_2 1}\quad p_{k_2 2}\right]\right\}_{k_2\in K_2(1)} \tag{15}$$

of nodes by a set of node indices $K_2(1) \subset \{\overline{2,\ N}\}$ in the second group, where

$$\begin{aligned}&K_1(1) \cup K_2(1) = \{\overline{1,\ N}\} \text{ and } K_1(1) \cap K_2(1) = \varnothing,\\ &\mathrm{N}_1(1) \cup \mathrm{N}_2(1) = \mathrm{N} \text{ and } \mathrm{N}_1(1) \cap \mathrm{N}_2(1) = \varnothing,\end{aligned} \tag{16}$$

while the depot factually defines (and will always define) cluster 1 (although there is no difference to which cluster the depot is assigned). This is step 1 of the clustering, which is denoted by the respective index in the brackets. In the subsequent clustering, the assignment of the depot affects the calculation, but does not affect the final result, i. e. any cluster can "take" the depot in. The final solution does not depend on the depot assignment, but the way to obtain the solution does depend on the depot assignment (Luo et al., 2022). If sets (14) and (15) are still intractably big to solve respective open-loop subproblems, they can be further divided into two subsets each. Set (14) is divided into non-overlapping subsets

$$\mathrm{N}_1(2) = \left\{\left[p_{k_1 1}\quad p_{k_1 2}\right]\right\}_{k_1\in K_1(2)} \tag{17}$$

by $K_1(2) \subset K_1(1)$, $1 \in K_1(2)$, and

$$\mathrm{N}_2(2) = \left\{\left[p_{k_2 1}\quad p_{k_2 2}\right]\right\}_{k_2\in K_2(2)} \tag{18}$$

by $K_2(2) \subset K_2(1)$, where

$$\begin{aligned}&K_1(2) \cup K_2(2) = K_1(1) \text{ and } K_1(2) \cap K_2(2) = \varnothing,\\ &\mathrm{N}_1(2) \cup \mathrm{N}_2(2) = \mathrm{N}_1(1) \text{ and } \mathrm{N}_1(2) \cap \mathrm{N}_2(2) = \varnothing,\end{aligned} \tag{19}$$

and set (15) is divided into non-overlapping subsets

$$\mathrm{N}_3(2) = \left\{\left[p_{k_3 1}\quad p_{k_3 2}\right]\right\}_{k_3\in K_3(2)} \tag{20}$$

by $K_3(2) \subset K_2(1)$, and

$$\mathrm{N}_4(2) = \left\{\left[p_{k_4 1}\quad p_{k_4 2}\right]\right\}_{k_4\in K_4(2)}, \tag{21}$$

by $K_4(2) \subset K_2(1)$, where

$$\begin{aligned}&K_3(2) \cup K_4(2) = K_2(1) \text{ and } K_3(2) \cap K_4(2) = \varnothing,\\ &\mathrm{N}_3(2) \cup \mathrm{N}_4(2) = \mathrm{N}_2(1) \text{ and } \mathrm{N}_3(2) \cap \mathrm{N}_4(2) = \varnothing.\end{aligned} \tag{22}$$

This is step 2 of the clustering, which is denoted by the respective index in the brackets. If sets (17), (18), (20), (21) are still intractably big to solve respective open-loop subproblems, they can be further divided into two subsets each. At step n of the clustering, $n \in \mathbb{N}\backslash\{1\}$, there are $M = 2^n$ clusters

$$\mathrm{N}_m(n) = \left\{\left[p_{k_m 1}\quad p_{k_m 2}\right]\right\}_{k_m\in K_m(n)} \text{ for } m = \overline{1,\ M} \tag{23}$$

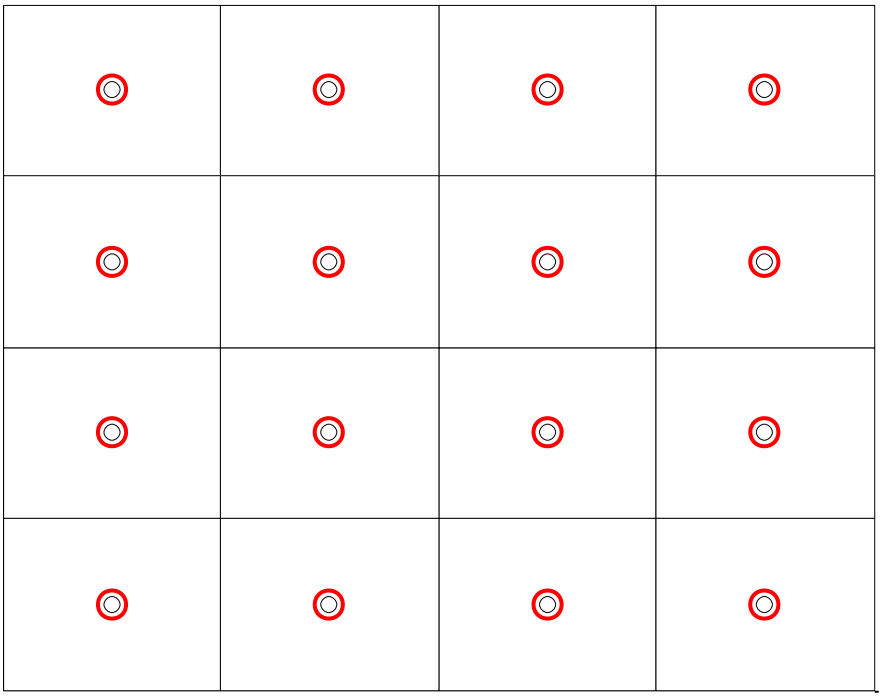


Fig. 1. The lattice of the clustering of the four steps depth.

by $K_1(n)\subset K_1(n-1)$, $1\in K_1(n)$, where

$$\begin{gathered}K_{2i-1}(n)\cup K_{2i}(n)=K_i(n-1)\ \text{ and }\ K_{2i-1}(n)\cap K_{2i}(n)=\varnothing,\\ \mathrm{N}_{2i-1}(n)\cup \mathrm{N}_{2i}(n)=\mathrm{N}_i(n-1)\ \text{ and }\ \mathrm{N}_{2i-1}(n)\cap \mathrm{N}_{2i}(n)=\varnothing\\ \text{for } i=\overline{1,\ 2^{n-1}}.\end{gathered}\tag{24}$$

Obviously, these clusters are non-overlapping, so

$$\begin{gathered}\bigcup_{m=1}^{M}K_m=\{\overline{1,\ N}\}\ \text{ and }\ \bigcup_{m=1}^{M}\mathrm{N}_m=\mathrm{N},\\ K_{m_1}\cap K_{m_2}=\varnothing\ \text{ for }\ m_1=\overline{1,\ M}\ \text{ and }\ m_2=\overline{1,\ M}\ \text{ by }\ m_1\neq m_2.\end{gathered}\tag{25}$$

As the clustering deepens (i. e., number n is increased), the clusters are broken quicker. The resulting clustering can be patterned as a rectangular lattice whose cell centers approximately correspond to the centroids

$$\{\mathbf{C}_m=[\,c_{m1}\quad c_{m2}\,]\}_{m=1}^{M}\tag{26}$$

of the clusters. An example of the lattice for the clustering of the four steps depth is presented in Fig. 1. An example of 16 real clusters is shown in Fig. 2. Surely, centroids of real clusters are only approximately close to the cell centers but generally do not coincide with them (Hartigan and Wong, 1979; Celebi et al., 2013; Ikotun et al., 2023).

Every cluster corresponds to its respective open-loop subproblem, whose solutions should be aggregated. Owing to the rectangular lattice consisting of 2^n cells, the aggregation is quite easy being a symmetric rectangular close-loop serpentine (Fig. 3). Before the aggregation, the clusters are re-numbered so that their new numbers correspond to the consecution of how the clusters are connected by the close-loop serpentine. This re-numbering can be done by determining which cluster is the closest to every cell center. The cluster containing the depot has always number 1, though. It is worth noting that the close-loop serpentines in Fig. 3 are not only possible. The other close-loop serpentines are easily obtained by simple geometrical transformations of those in Fig. 3. For instance, a 90-degree rotation of a close-loop serpentine makes another serpentine of the same length. So, other serpentines are not considered, because they would produce the same result.

For just two clusters (14) and (15), the aggregation is done almost trivially. No re-numbering is required in this case. The salesman departing from the depot must complete the open-loop subtour at a node of cluster (14) that is the farthest from the depot. On the other hand, this node must be the nearest to cluster (15) to resume building the route for the initial TSP via the shortest connection of the two open-loop subtours. Therefore, the first open-loop subtour destination node must have number

$$k_1^{*}\in\arg\max_{k_1\in K_1(1)}\rho(k_1,\ 1)\tag{27}$$

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

$$\{k_1^{**},\ k_2^{**}\}\in\arg\min_{k_1\in K_1(1),\ k_2\in K_2(1)}\rho(k_1,\ k_2).\tag{28}$$

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

$$\delta(k_1,\ k_2)=\max\{\rho(k_1,\ 1),\ \rho(k_1,\ k_2)\}-\rho(k_1,\ k_2),\tag{29}$$

are considered as a two-component vector

$$\mathbf{V}(k_1,\ k_2)=[\,\rho(k_1,\ 1)\quad \delta(k_1,\ k_2)\,].\tag{30}$$

Vector (30) contains the distance to the depot and the difference between maximum of distances in (27), (28) and the distance to the second cluster. Minimizing the latter according to (28) maximizes distance $\delta(k_1,\ k_2)$. Maximizing the distance to the depot according to (27) maximizes distance $\delta(k_1,\ k_2)$ also. This allows maximizing both distances in vector (30). The set V of such vectors is formed for all node indices $k_1\in K_1(1)$ and node indices $k_2\in K_2(1)$. A subset $\widetilde{V}\subset V$ is selected such that for every vector $\widetilde{\mathbf{V}}\left(\widetilde{k}_1,\ \widetilde{k}_2\right)\in\widetilde{V}$ and every vector $\mathbf{V}_0\left(k_1^{(0)},\ k_2^{(0)}\right)\in\left\{V\backslash\widetilde{V}\right\}$ either a pair of inequalities

$$\rho\left(\widetilde{k}_1,\ 1\right)\geqslant\rho\left(k_1^{(0)},\ 1\right)\ \text{ and }\ \delta\left(\widetilde{k}_1,\ \widetilde{k}_2\right)>\delta\left(k_1^{(0)},\ k_2^{(0)}\right)\tag{31}$$

or a pair of inequalities

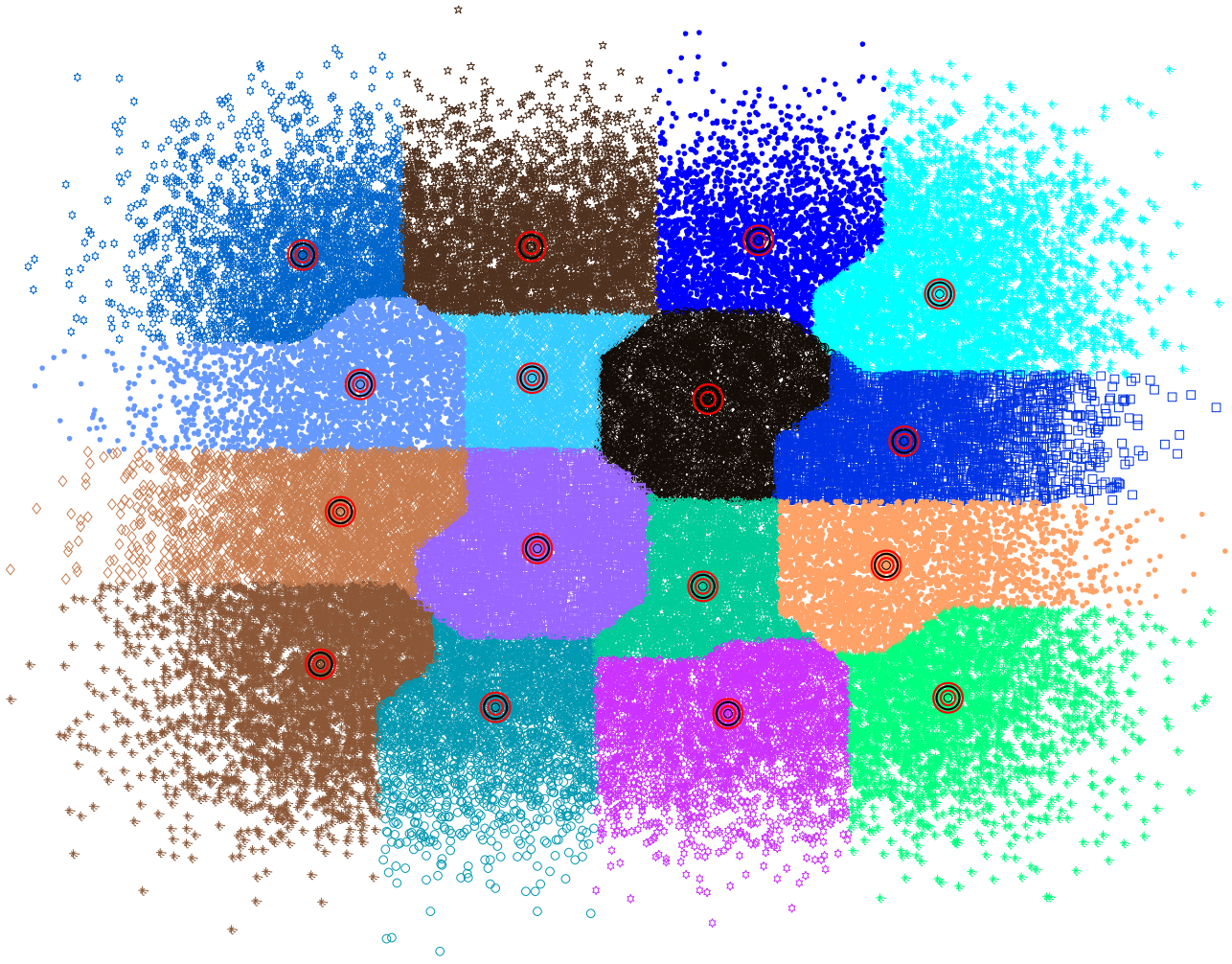


Fig. 2. 50,000 nodes divided into 16 clusters patterned as it is shown in Fig. 1.

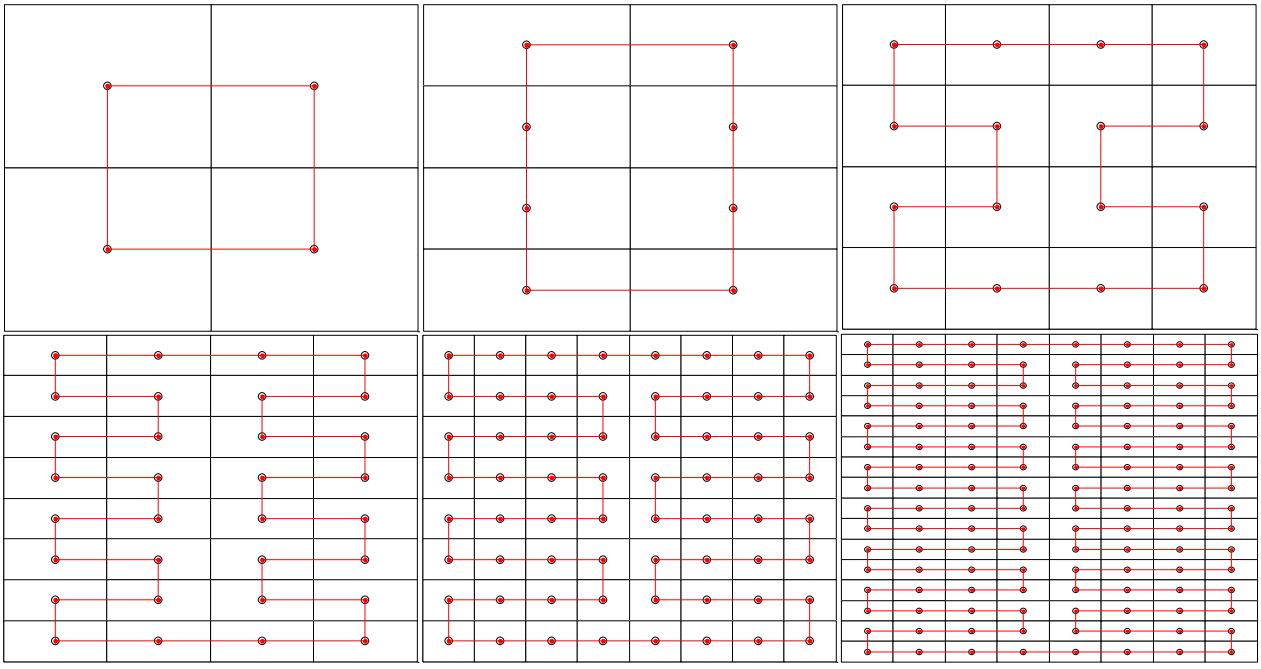


Fig. 3. The patterns of the shortest route passing through centers of the lattice cells.

$$\rho\left(\widetilde{k}_1,\ 1\right) > \rho\left(k_1^{(0)},\ 1\right) \text{ and } \delta\left(\widetilde{k}_1,\ \widetilde{k}_2\right) \geqslant \delta\left(k_1^{(0)},\ k_2^{(0)}\right) \tag{32}$$

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

$$\rho\left(\widetilde{k}_1^{(1)},\ 1\right) > \rho\left(\widetilde{k}_1^{(2)},\ 1\right) \text{ and } \delta\left(\widetilde{k}_1^{(1)},\ \widetilde{k}_2^{(1)}\right) < \delta\left(\widetilde{k}_1^{(2)},\ \widetilde{k}_2^{(2)}\right) \tag{33}$$

or a pair of inequalities

$$\rho\left(\widetilde{k}_1^{(1)},\ 1\right) < \rho\left(\widetilde{k}_1^{(2)},\ 1\right) \text{ and } \delta\left(\widetilde{k}_1^{(1)},\ \widetilde{k}_2^{(1)}\right) > \delta\left(\widetilde{k}_1^{(2)},\ \widetilde{k}_2^{(2)}\right) \tag{34}$$

is true, or the two inequalities

$$\rho\left(\widetilde{k}_1^{(1)},\ 1\right) = \rho\left(\widetilde{k}_1^{(2)},\ 1\right) \text{ and } \delta\left(\widetilde{k}_1^{(1)},\ \widetilde{k}_2^{(1)}\right) = \delta\left(\widetilde{k}_1^{(2)},\ \widetilde{k}_2^{(2)}\right) \tag{35}$$

are true. The lists of nodes in subset $\widetilde{V}\subset V$ are denoted by the respective subsets $\widetilde{K}_1(1)\subset K_1(1)$ and $\widetilde{K}_2(1)\subset K_2(1)$ of node indices. Then node $k_1^{***} \in \widetilde{K}_1(1)$ is selected such that the sum

$$\rho\left(k_1^{***},\ 1\right) + \delta\left(k_1^{***},\ k_2^{***}\right) \tag{36}$$

for some vector $\widetilde{\mathbf{V}}\left(k_1^{***},\ k_2^{***}\right) \in \widetilde{V}$ by $k_2^{***} \in \widetilde{K}_2$ 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 (23) — (25) for $n \in \mathbb{N}\backslash\{1\}$, the starting and destination nodes are determined similarly to (27) — (36). Node $k_{m-1}^{***} \in \widetilde{K}_{m-1}(n)$ 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 $k_m^{***} \in \widetilde{K}_m(n)$ for cluster m is selected such that the sum

$$\rho\left(k_m^{***},\ 1\right) + \delta\left(k_m^{***},\ k_{m+1}^{***}\right) \tag{37}$$

is maximal for some vector $\widetilde{\mathbf{V}}\left(k_m^{***},\ k_{m+1}^{***}\right) \in \widetilde{V}$, where $k_{m+1}^{***} \in \widetilde{K}_{m+1}(n)$, a set V of vectors

$$\mathbf{V}(k_m,\ k_{m+1}) = [\rho(k_m,\ 1) \quad \delta(k_m,\ k_{m+1})\ ] \tag{38}$$

is formed for all $k_m \in K_m(n)$ and $k_{m+1} \in K_{m+1}(n)$, a subset $\widetilde{V}\subset V$ by respective denotations $\widetilde{K}_m(n)\subset K_m(n)$ and $\widetilde{K}_{m+1}(n)\subset K_{m+1}(n)$ is selected such that for every vector $\widetilde{\mathbf{V}}\left(\widetilde{k}_m,\ \widetilde{k}_{m+1}\right) \in \widetilde{V}$ and every vector $\mathbf{V}_0\left(k_m^{(0)},\ k_{m+1}^{(0)}\right) \in \left\{V\backslash\widetilde{V}\right\}$ either a pair of inequalities

$$\rho\left(\widetilde{k}_m,\ 1\right) \geqslant \rho\left(k_m^{(0)},\ 1\right) \text{ and } \delta\left(\widetilde{k}_m,\ \widetilde{k}_{m+1}\right) > \delta\left(k_m^{(0)},\ k_{m+1}^{(0)}\right) \tag{39}$$

or a pair of inequalities

$$\rho\left(\widetilde{k}_m,\ 1\right) > \rho\left(k_m^{(0)},\ 1\right) \text{ and } \delta\left(\widetilde{k}_m,\ \widetilde{k}_{m+1}\right) \geqslant \delta\left(k_m^{(0)},\ k_{m+1}^{(0)}\right) \tag{40}$$

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

$$\rho\left(\widetilde{k}_m^{(1)},\ 1\right) > \rho\left(\widetilde{k}_m^{(2)},\ 1\right) \text{ and } \delta\left(\widetilde{k}_m^{(1)},\ \widetilde{k}_{m+1}^{(1)}\right) < \delta\left(\widetilde{k}_m^{(2)},\ \widetilde{k}_{m+1}^{(2)}\right) \tag{41}$$

or a pair of inequalities

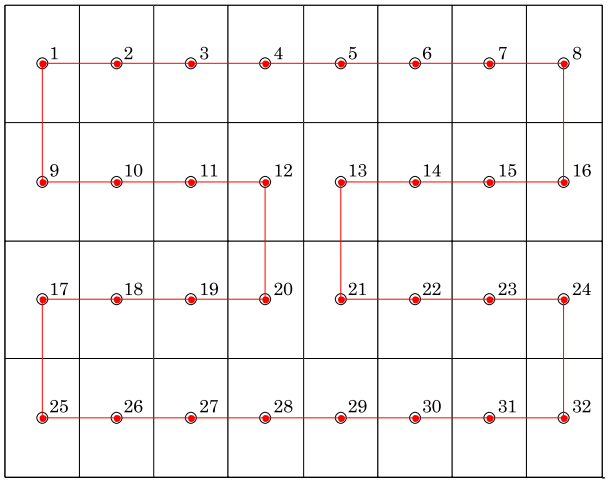


Fig. 4. The lattice of the clustering of the five steps depth (of eight horizontal cells) and the shortest route passing through the numbered centers of the lattice cells.

$$\rho\left(\widetilde{k}_m^{(1)},\ 1\right) < \rho\left(\widetilde{k}_m^{(2)},\ 1\right) \text{ and } \delta\left(\widetilde{k}_m^{(1)},\ \widetilde{k}_{m+1}^{(1)}\right) > \delta\left(\widetilde{k}_m^{(2)},\ \widetilde{k}_{m+1}^{(2)}\right) \tag{42}$$

is true, or the two inequalities

$$\rho\left(\widetilde{k}_m^{(1)},\ 1\right) = \rho\left(\widetilde{k}_m^{(2)},\ 1\right) \text{ and } \delta\left(\widetilde{k}_m^{(1)},\ \widetilde{k}_{m+1}^{(1)}\right) > \delta\left(\widetilde{k}_m^{(2)},\ \widetilde{k}_{m+1}^{(2)}\right) \tag{43}$$

are true. Node $k_{M-1}^{***} \in \widetilde{K}_{M-1}(n)$ 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.

Upon the clustering, the M open-loop subproblems can be solved in parallel (say, on parallel processor cores). Then, owing to $M = 2^n$, the subroutes (subtours) are efficiently aggregated through the depot and nodes $\left\{k_m^{***}\right\}_{m=1}^{M-1}$ making thus a closed loop route as an approximate solution to the initial TSP. 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 (Phillips, 2002; Gonzalez, 1985; Ikotun et al., 2023).

The cell centers are numbered starting from the left top corner downwards (see, e. g., Fig. 4). Initially, the clusters acquire their numbers as the clustering progresses deeper, and these numbers are most likely to differ from the numbers of the cell centers. Then the clusters are re-numbered in accordance with the numbers of the cell centers.

Having determined which cluster is the closest to every cell center, the consecution of how the clusters are connected by the close-loop serpentine is used to re-number the clusters. Then the shortest route passing through the re-numbered centroids is easily determined by using the following routine. If the lattice is of M_{hor} horizontal and M_{vert} vertical cells, where $M = M_{\text{hor}}\cdot M_{\text{vert}}$, and $\mathbf{U} = [u_m]_{1\times M}$ is the shortest route, then

$$u_m = m \quad \text{for} \quad m = \overline{1,\ M_{\text{hor}}}, \tag{44}$$

$$u_{mM_{\text{hor}}+q} = 2mM_{\text{hor}} - q + 1 \quad \text{for} \quad q = \overline{1,\ 0.5M_{\text{hor}}} \quad \text{and} \quad m = \overline{1,\ 0.5M_{\text{vert}} - 1}, \tag{45}$$

$$u_{mM_{\text{hor}}+0.5M_{\text{hor}}+q} = (2m+1)M_{\text{hor}} - 0.5M_{\text{hor}} + q \quad \text{for} \quad q = \overline{1,\ 0.5M_{\text{hor}}} \quad \text{and} \quad m = \overline{1,\ 0.5M_{\text{vert}} - 1}, \tag{46}$$

$$u_{0.5M+m} = M - m + 1 \quad \text{for} \quad m = \overline{1,\ M_{\text{hor}}}, \tag{47}$$

$$u_{0.5M+mM_{\text{hor}}+q} = (M_{\text{vert}} - 2m + 1)M_{\text{hor}} - M_{\text{hor}} + q \quad \text{for} \quad q = \overline{1,\ 0.5M_{\text{hor}}} \quad \text{and} \quad m = \overline{1,\ 0.5M_{\text{vert}} - 1} \tag{48}$$

$$u_{0.5M+mM_{\text{hor}}+0.5M_{\text{hor}}+q} = (M_{\text{vert}} - 2m)M_{\text{hor}} - 0.5M_{\text{hor}} - q + 1 \quad \text{for} \quad q = \overline{1,\ 0.5M_{\text{hor}}} \quad \text{and} \quad m = \overline{1,\ 0.5M_{\text{vert}} - 1}. \tag{49}$$

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

The initial set of nodes (13) belongs to rectangle

$$\left[\min_{k=\overline{1,\ N}} p_{k1};\ \max_{k=\overline{1,\ N}} p_{k1}\right] \times \left[\min_{k=\overline{1,\ N}} p_{k2};\ \max_{k=\overline{1,\ N}} p_{k2}\right]. \tag{50}$$

Rectangle (50) is uniformly broken into $M = 2^n$ cells (subrectangles)

$$\left[a_{m_{\text{hor}}m_{\text{vert}}};\ b_{m_{\text{hor}}m_{\text{vert}}}\right] \times \left[g_{m_{\text{hor}}m_{\text{vert}}};\ h_{m_{\text{hor}}m_{\text{vert}}}\right] = \left[\left(\min_{k=\overline{1,\ N}} p_{k1}\right) + (m_{\text{hor}} - 1)\cdot\frac{\max\limits_{k=\overline{1,\ N}} p_{k1} - \min\limits_{k=\overline{1,\ N}} p_{k1}}{M_{\text{hor}}};\ \left(\min_{k=\overline{1,\ N}} p_{k1}\right) + m_{\text{hor}}\cdot\frac{\max\limits_{k=\overline{1,\ N}} p_{k1} - \min\limits_{k=\overline{1,\ N}} p_{k1}}{M_{\text{hor}}}\right] \times \left[\left(\min_{k=\overline{1,\ N}} p_{k2}\right) + (m_{\text{vert}} - 1)\cdot\frac{\max\limits_{k=\overline{1,\ N}} p_{k2} - \min\limits_{k=\overline{1,\ N}} p_{k2}}{M_{\text{vert}}};\ \left(\min_{k=\overline{1,\ N}} p_{k2}\right) + m_{\text{vert}}\cdot\frac{\max\limits_{k=\overline{1,\ N}} p_{k2} - \min\limits_{k=\overline{1,\ N}} p_{k2}}{M_{\text{vert}}}\right] \tag{51}$$

for $m_{\text{hor}} = \overline{1,\ M_{\text{hor}}}$ and $m_{\text{vert}} = \overline{1,\ M_{\text{vert}}}$. The coordinates of the cell centers

$$\left\{\left\{\mathbf{R}_{m_{\text{hor}}m_{\text{vert}}} = \left[\, r_{m_{\text{hor}}m_{\text{vert}}1} \quad r_{m_{\text{hor}}m_{\text{vert}}2} \,\right]\right\}_{m_{\text{hor}}=1}^{M_{\text{hor}}}\right\}_{m_{\text{vert}}=1}^{M_{\text{vert}}} \tag{52}$$

are

$$\begin{aligned} r_{m_{\text{hor}}m_{\text{vert}}1} &= \frac{a_{m_{\text{hor}}m_{\text{vert}}} + b_{m_{\text{hor}}m_{\text{vert}}}}{2}, \\ r_{m_{\text{hor}}m_{\text{vert}}2} &= \frac{g_{m_{\text{hor}}m_{\text{vert}}} + h_{m_{\text{hor}}m_{\text{vert}}}}{2} \end{aligned} \tag{53}$$

for $m_{\text{hor}} = \overline{1,\ M_{\text{hor}}}$ and $m_{\text{vert}} = \overline{1,\ M_{\text{vert}}}$.

Each of centroids (26) should be referred to its subrectangle (cell). The set of these centroids can be thought of as a distorted lattice of size $M_{\text{vert}} \times M_{\text{hor}}$ (see Fig. 2 compared to Fig. 1). In fact, some centroids may not belong to their respective subrectangles (51), whereas some subrectangles may contain more than one centroid. Moreover, every centroid is not necessarily to be the uniquely closest to the respective cell center from set (52). To ascertain the centroid-to-cell reference (correspondence), it is sufficient to sort the centroids by $\{c_{m2}\}_{m=1}^{M}$ in descending order that gives M_{vert} rows of the lattice sorted by the ordinate component. Then every row centroids are sorted by the abscissa component in ascending order that gives a sequence of the numbers of the centroids corresponding to the numbers of the lattice cells. Finally, the shortest route passing through the centroids is given by the pattern in (44) — (49).

6. Genetic algorithm for TSP and open-loop subproblem

The genetic algorithm for solving a TSP requires such input parameters as a set of node locations (13), a population size, a set of mutation operators, and other auxiliary options that usually are set to their default values (Chambers, 2000; Haupt and Haupt, 2003). The population is a series of pseudorandom tours called chromosomes. Each element of the population is a route being a succession of nodes which the salesman should visit, where the population diversity, efficiency, and algorithm performance is prevented from declining by additional selection of the best chromosomes (Zheng et al., 2023).

For the genetic algorithm of solving an ordinary (close-loop) TSP, denote a succession of non-depot nodes, which the salesman should visit, by an $(N-1)$-dimensional vector

$$\mathbf{S} = [n_h]_{1\times(N-1)} \tag{54}$$

where

$$n_h \in \{\overline{2,\ N}\}\ \forall h = \overline{1,\ N-1}. \tag{55}$$

The salesman departs from the depot (node 1), which is not included into vector (54), and visits nodes n_1, n_2, …, n_{N-1} in this succession. Finally, the salesman departs from node n_{N-1} and arrives at the depot.

A population is generated as sequences of $N-1$ pseudorandom integers between 2 and N. For every route (54) 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

$$d = \rho(1,\ n_1) \tag{56}$$

and the remaining distances except the last one are accumulated into the running variable d:

$$d_{\text{obs}} = d, \quad d = d_{\text{obs}} + \rho(n_h,\ n_{h+1}) \quad \text{for} \quad h = \overline{1,\ N-2}. \tag{57}$$

The distance of returning to the depot is:

$$d_{\text{obs}} = d, \quad d = d_{\text{obs}} + \rho(n_{N-1},\ 1). \tag{58}$$

Then, sum

$$\widetilde{\rho}_{\Sigma}(\mathbf{S}) = d \geqslant \rho_{\Sigma}^{*} \tag{59}$$

is calculated and minimized over the population. Inequality (59) is the relationship between the length of a heuristically found route (54) and the shortest route length by an exact solution (there may be several different routes whose lengths are the same) given by (12) to problem (11).

A new population is generated using slide, flip, swap, and crossover operators (Chambers, 2000; Haupt and Haupt, 2003; Shafiee et al., 2016; Nagata and Kobayashi, 1999; Chen et al., 2012). The slide operator moves the last node from each chromosome to the beginning of another one. The flip operator swaps a random subsequence of nodes inside a chromosome: a subsequence

$$\left\{n_h^{(r)}\right\}_{h=h_1+1}^{h_2} \subset \left\{n_h^{(r)}\right\}_{h=1}^{N-1} \tag{60}$$

from a route

$$\mathbf{S}^{(r)} = \left[n_h^{(r)}\right]_{1\times(N-1)} \tag{61}$$

is extracted and flipped for pseudorandom indices h_1 and h_2 by

$$0 \leqslant h_1 < h_2 \leqslant N-1 \tag{62}$$

as

$$\mathbf{S}^{(r)}_{\text{obs}}=\left[n_h^{(r)*}\right]_{1\times(N-1)}=\mathbf{S}^{(r)},$$
$$\mathbf{S}^{(r)}=\left[n_h^{(r)}\right]_{1\times(N-1)}\ \text{ by }\ n_h^{(r)}=n_h^{(r)*}\ \forall h=\overline{1,\ h_1}\ \text{ and}$$
$$n_h^{(r)}=n^{(r)*}_{h_2+h_1-h+1}\ \forall h=\overline{h_1+1,\ h_2}\ \text{ and }\ n_h^{(r)}=n_h^{(r)*}\ \forall h=\overline{h_2+1,\ N-1}. \tag{63}$$

The flip operator returns then an updated vector (61) after (63).

For pseudorandom integers h_3 and h_4 by

$$0\leqslant h_3<h_4\leqslant N-1, \tag{64}$$

the swap operator selects the same-index-and-length subsequence of nodes from two chromosomes (61) and

$$\mathbf{S}^{(q)}=\left[n_h^{(q)}\right]_{1\times(N-1)} \tag{65}$$

for $r\neq q$ and interchange them: subsequences

$$\left\{n_h^{(r)}\right\}_{h=h_3+1}^{h_4}\subset\left\{n_h^{(r)}\right\}_{h=1}^{N-1} \tag{66}$$

and

$$\left\{n_h^{(q)}\right\}_{h=h_3+1}^{h_4}\subset\left\{n_h^{(q)}\right\}_{h=1}^{N-1} \tag{67}$$

are interchanged as

$$\begin{gathered}\mathbf{S}^{(r)}_{\text{obs}}=\left[n_h^{(r)*}\right]_{1\times(N-1)}=\mathbf{S}^{(r)},\\ \mathbf{S}^{(r)}=\left[n_h^{(r)}\right]_{1\times(N-1)}\ \text{ by }\ n_h^{(r)}=n_h^{(r)*}\ \forall h=\overline{1,\ h_3}\ \text{ and}\\ n_h^{(r)}=n_h^{(q)}\ \forall h=\overline{h_3+1,\ h_4}\ \text{ and }\ n_h^{(r)}=n_h^{(r)*}\ \forall h=\overline{h_4+1,\ N-1}\end{gathered} \tag{68}$$

and

$$\begin{gathered}\mathbf{S}^{(q)}_{\text{obs}}=\left[n_h^{(q)*}\right]_{1\times(N-1)}=\mathbf{S}^{(q)},\\ \mathbf{S}^{(q)}=\left[n_h^{(q)}\right]_{1\times(N-1)}\ \text{ by }\ n_h^{(q)}=n_h^{(q)*}\ \forall h=\overline{1,\ h_3}\ \text{ and}\\ n_h^{(q)}=n_h^{(r)*}\ \forall h=\overline{h_3+1,\ h_4}\ \text{ and }\ n_h^{(q)}=n_h^{(q)*}\ \forall h=\overline{h_4+1,\ N-1}.\end{gathered} \tag{69}$$

The swap operator returns then updated vectors (61) and (65) after (68) and (69), respectively.

The crossover operator, somewhat resembling the swapping, takes two chromosomes (61) and (65) for $r\neq q$, and cuts each chromosome in two random parts using pseudorandom integers h_r and h_q by $1\leqslant h_r\leqslant N-2$ and $1\leqslant h_q\leqslant N-2$, where h_r first nodes in chromosome (61) are left and h_q first nodes in chromosome (65) are left. Thereupon the chromosome parts of H nodes long by

$$1\leqslant H\leqslant N-1-\max\left\{h_r,\ h_q\right\}$$

are interchanged:

$$\begin{gathered}\mathbf{S}^{(r)**}=\left[n_h^{(r)**}\right]_{1\times(N-1)}\\ =\left\{\left\{n_h^{(r)}\right\}_{h=1}^{h_r},\ \left\{n_h^{(q)}\right\}_{h=h_q+1}^{h_q+H},\ \left\{n_h^{(r)}\right\}_{h=h_r+H+1}^{N-1}\right\}\end{gathered} \tag{70}$$

and

$$\begin{gathered}\mathbf{S}^{(q)**}=\left[n_h^{(q)**}\right]_{1\times(N-1)}\\ =\left\{\left\{n_h^{(q)}\right\}_{h=1}^{h_q},\ \left\{n_h^{(r)}\right\}_{h=h_r+1}^{h_r+H},\ \left\{n_h^{(q)}\right\}_{h=h_q+H+1}^{N-1}\right\}.\end{gathered} \tag{71}$$

The crossover operator has linear runtime complexity (Tinós et al., 2020).

The algorithm for solving the open-loop TSP is slightly modified: instead of route (54), a succession of non-start–end nodes, which the salesman should visit, is denoted by an $(N-2)$-dimensional vector

$$\mathbf{S}=[n_h]_{1\times(N-2)} \tag{72}$$

where

$$n_h\in\{\overline{2,\ N-1}\}\ \ \forall h=\overline{1,\ N-2}. \tag{73}$$

The salesman departed from the depot visits nodes $n_1, n_2, \ldots, n_{N-2}$ in this succession. Finally, the salesman departs from node n_{N-2} and arrives at node n_{N-1} which is the destination node. The distance to the node following the depot is calculated as (56) and the remaining distances except the last one are accumulated into the running variable d:

$$d_{\text{obs}}=d,\quad d=d_{\text{obs}}+\rho(n_h,\ n_{h+1})\ \text{ for }\ h=\overline{1,\ N-3}. \tag{74}$$

The distance to node N is:

$$d_{\text{obs}}=d,\quad d=d_{\text{obs}}+\rho(n_{N-2},\ N). \tag{75}$$

Then, sum (59) is calculated and minimized over the population. Instead of (60) — (71), a new population is similarly generated by

$$\left\{n_h^{(r)}\right\}_{h=h_1+1}^{h_2}\subset\left\{n_h^{(r)}\right\}_{h=1}^{N-2}, \tag{76}$$

$$\mathbf{S}^{(r)}=\left[n_h^{(r)}\right]_{1\times(N-2)}, \tag{77}$$

$$0\leqslant h_1<h_2\leqslant N-2, \tag{78}$$

$$\begin{gathered}\mathbf{S}^{(r)}_{\text{obs}}=\left[n_h^{(r)*}\right]_{1\times(N-2)}=\mathbf{S}^{(r)},\\ \mathbf{S}^{(r)}=\left[n_h^{(r)}\right]_{1\times(N-2)}\ \text{ by }\ n_h^{(r)}=n_h^{(r)*}\ \forall h=\overline{1,\ h_1}\ \text{ and}\\ n_h^{(r)}=n^{(r)*}_{h_2+h_1-h+1}\ \forall h=\overline{h_1+1,\ h_2}\ \text{ and }\ n_h^{(r)}=n_h^{(r)*}\ \forall h=\overline{h_2+1,\ N-2},\end{gathered} \tag{79}$$

$$0\leqslant h_3<h_4\leqslant N-2, \tag{80}$$

$$\mathbf{S}^{(q)}=\left[n_h^{(q)}\right]_{1\times(N-2)}, \tag{81}$$

$$\left\{n_h^{(r)}\right\}_{h=h_3+1}^{h_4}\subset\left\{n_h^{(r)}\right\}_{h=1}^{N-2}, \tag{82}$$

$$\left\{n_h^{(q)}\right\}_{h=h_3+1}^{h_4}\subset\left\{n_h^{(q)}\right\}_{h=1}^{N-2}, \tag{83}$$

$$\begin{gathered}\mathbf{S}^{(r)}_{\text{obs}}=\left[n_h^{(r)*}\right]_{1\times(N-2)}=\mathbf{S}^{(r)},\\ \mathbf{S}^{(r)}=\left[n_h^{(r)}\right]_{1\times(N-2)}\ \text{ by }\ n_h^{(r)}=n_h^{(r)*}\ \forall h=\overline{1,\ h_3}\ \text{ and}\\ n_h^{(r)}=n_h^{(q)}\ \forall h=\overline{h_3+1,\ h_4}\ \text{ and }\ n_h^{(r)}=n_h^{(r)*}\ \forall h=\overline{h_4+1,\ N-2},\end{gathered} \tag{84}$$

$$\begin{gathered}\mathbf{S}^{(q)}_{\text{obs}}=\left[n_h^{(q)*}\right]_{1\times(N-2)}=\mathbf{S}^{(q)},\\ \mathbf{S}^{(q)}=\left[n_h^{(q)}\right]_{1\times(N-2)}\ \text{ by }\ n_h^{(q)}=n_h^{(q)*}\ \forall h=\overline{1,\ h_3}\ \text{ and}\\ n_h^{(q)}=n_h^{(r)*}\ \forall h=\overline{h_3+1,\ h_4}\ \text{ and }\ n_h^{(q)}=n_h^{(q)*}\ \forall h=\overline{h_4+1,\ N-2},\end{gathered} \tag{85}$$

$$\begin{gathered}\mathbf{S}^{(r)**}=\left[n_h^{(r)**}\right]_{1\times(N-2)}\\ =\left\{\left\{n_h^{(r)}\right\}_{h=1}^{h_r},\ \left\{n_h^{(q)}\right\}_{h=h_q+1}^{h_q+H},\ \left\{n_h^{(r)}\right\}_{h=h_r+H+1}^{N-2}\right\}\end{gathered} \tag{86}$$

and

$$\begin{gathered}\mathbf{S}^{(q)**}=\left[n_h^{(q)**}\right]_{1\times(N-2)}\\ =\left\{\left\{n_h^{(q)}\right\}_{h=1}^{h_q},\ \left\{n_h^{(r)}\right\}_{h=h_r+1}^{h_r+H},\ \left\{n_h^{(q)}\right\}_{h=h_q+H+1}^{N-2}\right\}\end{gathered} \tag{87}$$

by

$$1 \leqslant h_r \leqslant N-3,$$

$$1 \leqslant h_q \leqslant N-3,$$

$$1 \leqslant H \leqslant N-2-\max\{h_r,\ h_q\},$$

respectively. Henceforward, the genetic algorithm is run by (56) — (59) and (60) — (71) for solving the whole TSP, and it is run by (56), (74), (75), (59) and (76) — (87) for solving the open-loop TSP.

7. Parallelization efficiency

To see how much building open-loop subroutes in parallel and aggregating them by the symmetric rectangular close-loop serpentine as (44) — (49) is faster than solving the whole TSP, without losing much in accuracy, the performances in the computational time and route length of the two approaches must be compared. In addition, the open-loop subroutes can be aggregated by the shortest closed loop route passing through centroids (26). This route is found as the respective TSP solution. In further consideration, let this supplementary TSP be called the centroid TSP.

Initially, the number of nodes is varied from 1751 to 7001:

$$N \in \{1751,\ 3501,\ 5251,\ 7001\}. \tag{88}$$

Let three versions of the clustering depth (clustering of the two, four, and six steps depth) be used:

$$M \in \{4,\ 16,\ 64\}. \tag{89}$$

To randomly generate nodes, pseudorandom numbers independently drawn from the standard uniform distribution on the open interval $(0;\ 1)$ are used (Matsumoto and Nishimura, 1998; Kneusel, 2018). Denote them by θ_1, θ_2. Besides, pseudorandom numbers independently drawn from the standard normal distribution are used. Denote them by η_1, η_2, η_3, η_4. The node locations are generated as

$$\begin{aligned} p_{k1} &= 100\cdot\theta_1 + 10\cdot\eta_1 + 50 \text{ and} \\ p_{k2} &= 100\cdot\theta_2 + 10\cdot\eta_2 + 50 \text{ by } k = \overline{2,\ N}. \end{aligned} \tag{90}$$

The depot location is

$$\begin{aligned} p_{11} &= \frac{1}{N-1}\cdot\sum_{k=2}^{N} p_{k1} + 10\cdot\eta_3 \text{ and} \\ p_{12} &= \frac{1}{N-1}\cdot\sum_{k=2}^{N} p_{k2} + 10\cdot\eta_4. \end{aligned} \tag{91}$$

Denote the number of nodes in cluster m by N_m, where the depot in cluster 1 is counted separately. So, $N = 1 + \sum_{m=1}^{M} N_m$. The maximal number of iterations for solving the whole TSP is set to $200\cdot(N-1)$, and the maximal number of iterations for solving the open-loop subproblems is similarly set to $200\cdot N_m$. 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. This is similar to using a search limitation strategy to speed up the local search phase (Rocha and Subramanian, 2023). To obtain reliable and stable statistical data, the whole TSP is re-generated five times for every number of nodes (88) and clustering depth (89).

Nodes (13) are clustered into M groups by using the K-means method (Hartigan and Wong, 1979; Celebi et al., 2013; Ikotun et al., 2023), according to which a subset $\mathrm{N}_{AB} \subset \mathrm{N}$ of nodes $\mathbf{N}_q$ is divided into two groups A and B. At the first iteration of the two-group division, centroids $\mathbf{C}_A(1) \in \mathrm{N}_{AB}$ and $\mathbf{C}_B(1) \in \mathrm{N}_{AB}$ are randomly assigned by $\mathbf{C}_A(1) \neq \mathbf{C}_B(1)$. At iteration t of the two-group division, the groups are determined as

$$A(t) = \left\{\mathbf{N}_q \in \mathrm{N}_{AB} : \rho_{\mathbb{R}^2}\left(\mathbf{N}_q,\ \mathbf{C}_A(t)\right) \leqslant \rho_{\mathbb{R}^2}\left(\mathbf{N}_q,\ \mathbf{C}_B(t)\right)\right\}, \tag{92}$$

$$B(t) = \left\{\mathbf{N}_q \in \mathrm{N}_{AB} : \rho_{\mathbb{R}^2}\left(\mathbf{N}_q,\ \mathbf{C}_B(t)\right) \leqslant \rho_{\mathbb{R}^2}\left(\mathbf{N}_q,\ \mathbf{C}_A(t)\right)\right\} \tag{93}$$

for updating their centroids

$$\mathbf{C}_A(t+1) = \frac{1}{|A(t)|}\cdot\sum_{\mathbf{N}_q \in A(t)} \mathbf{N}_q\ , \tag{94}$$

$$\mathbf{C}_B(t+1) = \frac{1}{|B(t)|}\cdot\sum_{\mathbf{N}_q \in B(t)} \mathbf{N}_q\ , \tag{95}$$

where $t = 1,\ 2,\ 3,\ \ldots$, by the distance $\rho_{\mathbb{R}^2}$ calculated similarly to distance (1). If occasionally

$$\rho_{\mathbb{R}^2}\left(\mathbf{N}_q,\ \mathbf{C}_B(t)\right) = \rho_{\mathbb{R}^2}\left(\mathbf{N}_q,\ \mathbf{C}_A(t)\right)$$

then node $\mathbf{N}_q$ is randomly assigned to either group (92) or group (93), but not to both. The alternating calculations (92) — (95) are run until

$$\rho_{\mathbb{R}^2}(\mathbf{C}_A(t^*+1),\ \mathbf{C}_A(t^*)) < \varepsilon \tag{96}$$

and

$$\rho_{\mathbb{R}^2}(\mathbf{C}_B(t^*+1),\ \mathbf{C}_B(t^*)) < \varepsilon \tag{97}$$

for a sufficiently small $\varepsilon > 0$ at some iteration t^*. Then the two-group division results in groups $A(t^*)$ and $B(t^*)$ whose centroids are $\mathbf{C}_A(t^*)$ and $\mathbf{C}_B(t^*)$, respectively. The two-group divisions by (92) — (97) can be run in parallel also at every step of the clustering.

For the case of the symmetric rectangular close-loop serpentine, denote the shortest length of the subroute in the subproblem for cluster m by $\widetilde{\rho}_{\Sigma}^{\exists *(m)}(N_m,\ M;\ w)$ for the w-th whole TSP generated for N nodes by (88) and M clusters by (89), $w = \overline{1,\ 5}$. For the case of the centroid TSP, denote the shortest length of the subroute in the subproblem for cluster m by $\widetilde{\rho}_{\Sigma}^{(\mathbf{C})*(m)}(N_m,\ M;\ w)$. Denote by $\widetilde{\rho}_{\Sigma}^{*}(N;\ w)$ the shortest route length found for the w-th whole TSP generated for N nodes by (88). The two comparisons of the parallelization gain are to be made. First, the route gain of parallelizing the whole TSP by using the symmetric rectangular close-loop serpentine is estimated by a ratio

$$g(N,\ M;\ w) = \frac{\widetilde{\rho}_{\Sigma}^{*}(N;\ w)}{\sum_{m=1}^{M}\widetilde{\rho}_{\Sigma}^{\exists *(m)}(N_m,\ M;\ w)}. \tag{98}$$

If $g(N,\ M;\ w) > 1$ then it means that the length of the route made from aggregating the serpentine-based subroutes from the M clusters is shorter than that length $\widetilde{\rho}_{\Sigma}^{*}(N;\ w)$ obtained without parallelization. If $g(N,\ M;\ w) < 1$ then the parallelization worsens the performance for the given TSP. Second, the route gain of parallelizing the whole TSP by using the serpentine-based subroutes versus using the centroid-based subroutes is estimated by a ratio

$$f(N,\ M;\ w) = \frac{\sum_{m=1}^{M}\widetilde{\rho}_{\Sigma}^{(\mathbf{C})*(m)}(N_m,\ M;\ w)}{\sum_{m=1}^{M}\widetilde{\rho}_{\Sigma}^{\exists *(m)}(N_m,\ M;\ w)}. \tag{99}$$

If $f(N,\ M;\ w) > 1$ then it means that the length of the route aggregated from the serpentine-based M subroutes is shorter than the length of the route aggregated from the centroid-based M subroutes. If $f(N,\ M;\ w) < 1$ then the serpentine-based parallelization is worse than the centroid-based one for the given TSP.

In the case $f(N,\ M;\ w) = 1$ the serpentine-based parallelization still works because the centroid-based parallelization additionally spends computational time to solve the centroid TSP. Denote by $l_{\Sigma}^{*}(N;\ w)$ the number of iterations taken to solve the whole TSP, denote by $l_{\Sigma}^{(\mathbf{C})*}(M;\ w)$ the number of iterations taken to solve the centroid TSP, denote by $l_{\Sigma}^{(\mathbf{C})*(m)}(N_m,\ M;\ w)$ the number of iterations taken to solve centroid-based subproblem m, and denote by $l_{\Sigma}^{\exists *(m)}(N_m,\ M;\ w)$ the number of iterations

Table 1
Statistics (104) — (109) of route gains (98) and (99) for parallelizing the TSPs generated by (88) — (91).

N	$g_{\min}(N,\ M)$			$\overline{g}(N,\ M)$			$g_{\max}(N,M)$		
	$M\ =\ 4$	$M=16$	$M=64$	$M=4$	$M=16$	$M=64$	$M=4$	$M=16$	$M=64$
1751	0.9978	0.97	0.9233	**1.0047**	0.9875	0.9331	**1.0278**	**1.009**	0.9398
3501	**1.0468**	**1.0272**	0.9759	**1.0522**	**1.0398**	0.9937	**1.0602**	**1.0479**	**1.0034**
5251	**1.1134**	**1.1069**	**1.0637**	**1.1238**	**1.1127**	**1.0754**	**1.1356**	**1.12**	**1.0911**
7001	**1.1916**	**1.1808**	**1.1424**	**1.1958**	**1.1849**	**1.1524**	**1.2028**	**1.1901**	**1.1696**

N	$f_{\min}(N,\ M)$			$\overline{f}(N,\ M)$			$f_{\max}(N,M)$		
	$M\ =\ 4$	$M=16$	$M=64$	$M=4$	$M=16$	$M=64$	$M=4$	$M=16$	$M=64$
1751	**1.0027**	0.9949	0.9866	**1.0105**	**1.0062**	0.9906	**1.0252**	**1.015**	0.9936
3501	0.9863	0.9959	0.9759	0.9976	**1.0046**	0.9932	**1.0092**	**1.0151**	**1.0013**
5251	0.9891	0.9995	0.9861	**1.0001**	**1.0051**	0.9918	**1.0089**	**1.0089**	0.9996
7001	0.9954	0.9917	0.9911	**1.0027**	0.9992	0.9975	**1.0068**	**1.0052**	**1.0036**

taken to solve serpentine-based subproblem m. Then the speed gain of parallelizing the whole TSP by using the symmetric rectangular close-loop serpentine is estimated by a ratio

$$\lambda(N,\ M;\ w)=\frac{l_{\Sigma}^{*}(N;\ w)}{\sum_{m=1}^{M}\overline{l}_{\Sigma}^{\exists *(m)}(N_m,\ M;\ w)}. \tag{100}$$

The speed gain of parallelizing the whole TSP by using the serpentine-based subroutes versus using the centroid-based subroutes is estimated by a ratio

$$\mu(N,\ M;\ w)=\frac{l_{\Sigma}^{*(\mathbf{C})}(M;\ w)+\sum_{m=1}^{M}l_{\Sigma}^{(\mathbf{C})*(m)}(N_m,\ M;\ w)}{\sum_{m=1}^{M}\overline{l}_{\Sigma}^{\exists *(m)}(N_m,\ M;\ w)}. \tag{101}$$

Each of ratios (100) and (101) implies that the subproblems are not solved in parallel, though. When they are solved in parallel, then the speed gains are:

$$\lambda^{(\mathrm{par})}(N,\ M;\ w)=\frac{l_{\Sigma}^{*}(N;\ w)}{\max\limits_{m=\overline{1,\ M}}\overline{l}_{\Sigma}^{\exists *(m)}(N_m,\ M;\ w)}, \tag{102}$$

$$\mu^{(\mathrm{par})}(N,\ M;\ w)=\frac{l_{\Sigma}^{*(\mathbf{C})}(M;\ w)+\max\limits_{m=\overline{1,\ M}}l_{\Sigma}^{(\mathbf{C})*(m)}(N_m,\ M;\ w)}{\max\limits_{m=\overline{1,\ M}}\overline{l}_{\Sigma}^{\exists *(m)}(N_m,\ M;\ w)}. \tag{103}$$

If $\lambda(N,\ M;\ w)>1$ then it means that, even when the M subproblems are solved sequentially, the aggregated serpentine-based solution is obtained faster than a solution of the whole TSP. If $\lambda^{(\mathrm{par})}(N,\ M;\ w)>1$ then it means that the aggregated serpentine-based solution is obtained faster by solving the M subproblems in parallel. If $\mu(N,\ M;\ w)>1$ then it means that, even when the M subproblems are solved sequentially, the serpentine-based parallelization is faster than the centroid-based one for the given TSP. In other words, here the sequential parallelization speed gain exists with respect to the centroid-based parallelization. If $\mu^{(\mathrm{par})}(N,\ M;\ w)>1$ then it means that the serpentine-based parallelization is faster than the centroid-based one when the M subproblems are solved in parallel, i. e. the non-sequential parallelization speed gain exists with respect to the centroid-based parallelization.

To study the mini-statistics of the parallelization gains by (98) — (103), their minimal, average, and maximal values calculated respectively as

$$g_{\min}(N,\ M)=\min_{w=\overline{1,\ 5}}g(N,\ M;\ w), \tag{104}$$

$$\overline{g}(N,\ M)=\frac{1}{5}\cdot\sum_{w=1}^{5}g(N,\ M;\ w), \tag{105}$$

$$g_{\max}(N,\ M)=\max_{w=\overline{1,\ 5}}g(N,\ M;\ w), \tag{106}$$

$$f_{\min}(N,\ M)=\min_{w=\overline{1,\ 5}}f(N,\ M;\ w), \tag{107}$$

$$\overline{f}(N,\ M)=\frac{1}{5}\cdot\sum_{w=1}^{5}f(N,\ M;\ w), \tag{108}$$

$$f_{\max}(N,\ M)=\max_{w=\overline{1,\ 5}}f(N,\ M;\ w), \tag{109}$$

$$\lambda_{\min}(N,\ M)=\min_{w=\overline{1,\ 5}}\lambda(N,\ M;\ w), \tag{110}$$

$$\overline{\lambda}(N,\ M)=\frac{1}{5}\cdot\sum_{w=1}^{5}\lambda(N,\ M;\ w), \tag{111}$$

Table 2
Statistics (110) — (115) of speed gains (100) and (101) for parallelizing the TSPs generated by (88) — (91).

N	$\lambda_{\min}(N,\ M)$			$\overline{\lambda}(N,\ M)$			$\lambda_{\max}(N,M)$		
	$M\ =\ 4$	$M=16$	$M=64$	$M=4$	$M=16$	$M=64$	$M=4$	$M=16$	$M=64$
1751	**1.7966**	**4.9358**	**8.05**	**2.1452**	**5.2105**	**8.1452**	**2.4701**	**5.3868**	**8.2015**
3501	**1.151**	**3.2493**	**6.8251**	**1.1868**	**3.47**	**6.8911**	**1.2492**	**3.682**	**6.9866**
5251	1	**2.5702**	**5.7619**	**1.003**	**2.6263**	**5.9337**	**1.0149**	**2.7**	**6.1535**
7001	1	**1.9414**	**4.9622**	1	**2.0761**	**5.1418**	1	**2.1958**	**5.3362**

N	$\mu_{\min}(N,\ M)$			$\overline{\mu}(N,\ M)$			$\mu_{\max}(N,M)$		
	$M\ =\ 4$	$M=16$	$M=64$	$M=4$	$M=16$	$M=64$	$M=4$	$M=16$	$M=64$
1751	0.851	0.9558	**1.03**	**1.0802**	0.9923	**1.0467**	**1.2564**	**1.0352**	**1.0561**
3501	0.9635	0.9595	**1.0074**	**1.0082**	0.9989	**1.0142**	**1.0931**	**1.0615**	**1.0252**
5251	0.9882	0.9648	0.9973	0.9978	0.9878	**1.0161**	**1.0089**	**1.0243**	**1.0356**
7001	**1.0001**	0.9143	0.9585	**1.0001**	**1.0194**	**1.004**	**1.0001**	**1.1377**	**1.0272**

Table 3
Statistics (116)—(121) of speed gains (102) and (103) for parallelizing the TSPs generated by (88) — (91).

N	$\lambda_{\min}^{(\mathrm{par})}(N,\ M)$			$\overline{\lambda}^{(\mathrm{par})}(N,\ M)$			$\lambda_{\max}^{(\mathrm{par})}(N,M)$		
	$M\ =\ 4$	$M\ =16$	$M\ =64$	$M\ =4$	$M\ =16$	$M\ =64$	$M\ =4$	$M\ =16$	$M\ =64$
1751	**4.6105**	**31.6857**	**185.8736**	**6.5765**	**42.3974**	**205.7855**	**8.2526**	**47.2271**	**218.3406**
3501	**3.3884**	**26.1467**	**119.0476**	**3.7513**	**31.0313**	**186.9617**	**4.2385**	**37.0331**	**225.6609**
5251	**3.4135**	**20.2032**	**145.1479**	**3.6344**	**22.6522**	**164.3746**	**3.8688**	**25.2264**	**199.5818**
7001	**3.4843**	**16.0013**	**90.3984**	**3.6733**	**18.7954**	**124.9446**	**3.937**	**22.7376**	**154.9701**

N	$\mu_{\min}^{(\mathrm{par})}(N,\ M)$			$\overline{\mu}^{(\mathrm{par})}(N,\ M)$			$\mu_{\max}^{(\mathrm{par})}(N,M)$		
	$M\ =\ 4$	$M\ =16$	$M\ =64$	$M\ =4$	$M\ =16$	$M\ =64$	$M\ =4$	$M\ =16$	$M\ =64$
1751	0.6696	0.858	**1.9857**	**1.0984**	0.957	**2.0976**	**1.4106**	**1.0777**	**2.2177**
3501	0.9172	0.8863	**1.1633**	0.9757	0.9843	**1.4202**	**1.0342**	**1.0533**	**1.5711**
5251	**1.0003**	0.9721	**1.146**	**1.0003**	**1.0563**	**1.3948**	**1.0003**	**1.1637**	**1.6136**
7001	**1.0002**	0.8687	0.7316	**1.0002**	**1.1008**	**1.1011**	**1.0002**	**1.4168**	**1.3607**

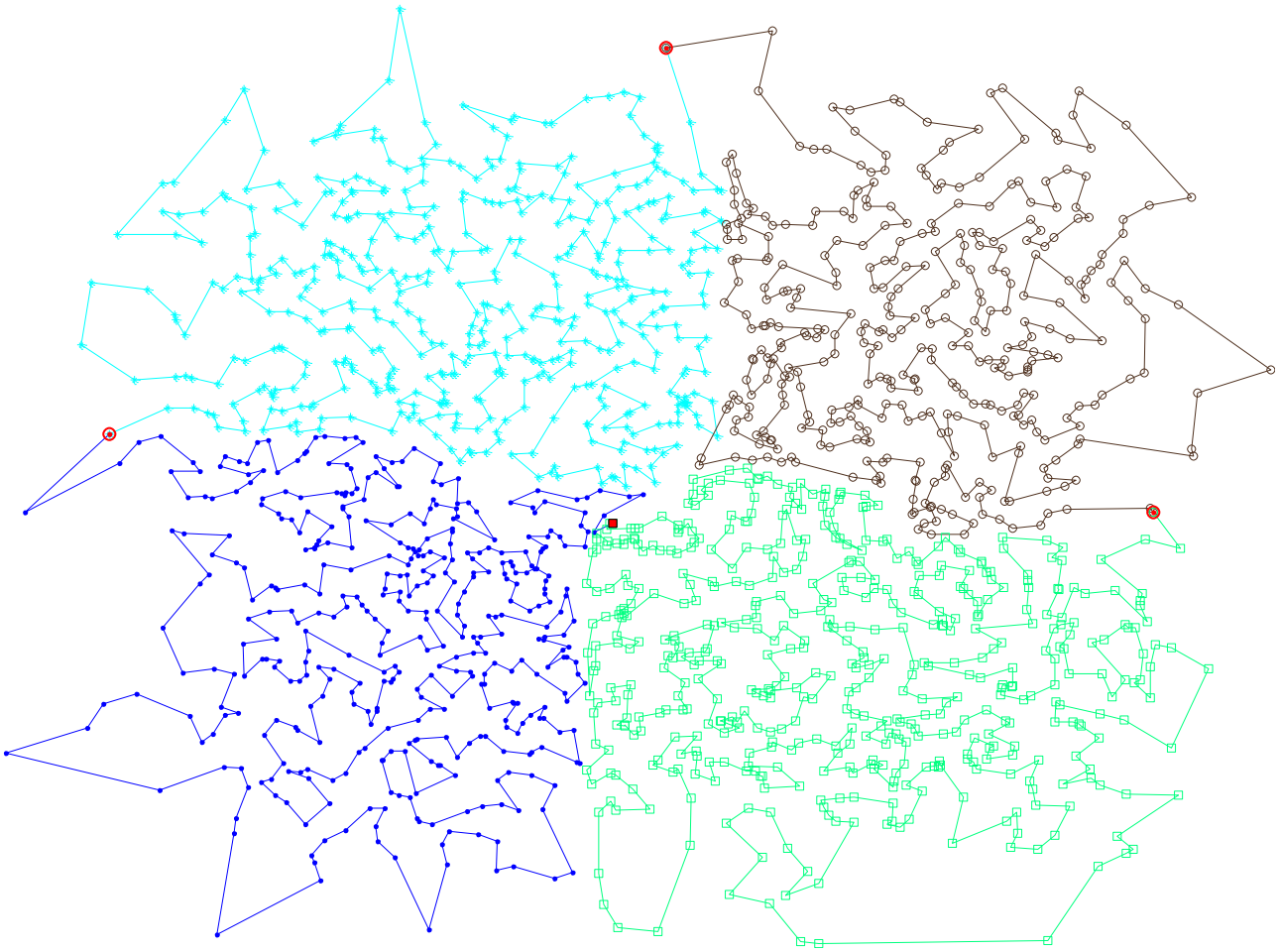


Fig. 5. The approximately shortest route aggregated clockwise from the four open-loop subroutes (whose respective lengths are 1075.2276, 1156.4366, 1008.4883, 1260.5488) by applying the centroid-based approach of the parallelization.

$$\lambda_{\max}(N,\ M)=\max_{w=\overline{1,\ 5}}\lambda(N,\ M;\ w), \tag{112}$$

$$\mu_{\min}(N,\ M)=\min_{w=\overline{1,\ 5}}\mu(N,\ M;\ w), \tag{113}$$

$$\overline{\mu}(N,\ M)=\frac{1}{5}\cdot\sum_{w=1}^{5}\mu(N,\ M;\ w), \tag{114}$$

$$\mu_{\max}(N,\ M)=\max_{w=\overline{1,\ 5}}\mu(N,\ M;\ w), \tag{115}$$

$$\lambda_{\min}^{(\mathrm{par})}(N,\ M)=\min_{w=\overline{1,\ 5}}\lambda^{(\mathrm{par})}(N,\ M;\ w), \tag{116}$$

$$\overline{\lambda}^{(\mathrm{par})}(N,\ M)=\frac{1}{5}\cdot\sum_{w=1}^{5}\lambda^{(\mathrm{par})}(N,\ M;\ w), \tag{117}$$

$$\lambda_{\max}^{(\mathrm{par})}(N,\ M)=\max_{w=\overline{1,\ 5}}\lambda^{(\mathrm{par})}(N,\ M;\ w), \tag{118}$$

$$\mu_{\min}^{(\mathrm{par})}(N,\ M)=\min_{w=\overline{1,\ 5}}\mu^{(\mathrm{par})}(N,\ M;\ w), \tag{119}$$

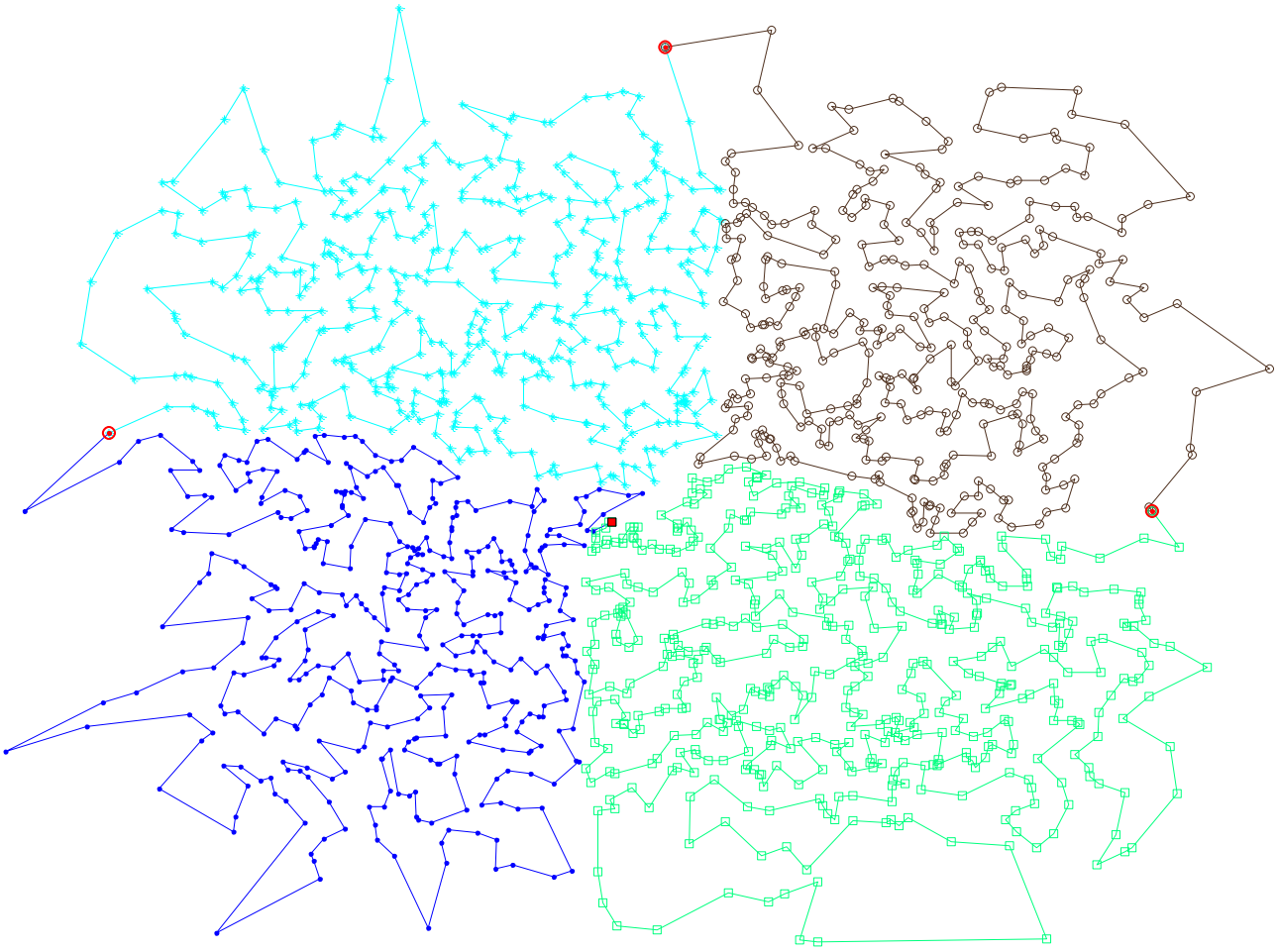


Fig. 6. The approximately shortest route aggregated clockwise from the four open-loop subroutes (whose respective lengths are 1067.847, 1145.4293, 983.11448, 1238.528 being each less than those in Fig. 5) by applying the serpentine-based approach of the parallelization.

$$\overline{\mu}^{(\mathrm{par})}(N,\ M)=\frac{1}{5}\cdot\sum_{w=1}^{5}\mu^{(\mathrm{par})}(N,\ M;\ w), \tag{120}$$

$$\mu_{\max}^{(\mathrm{par})}(N,\ M)=\max_{w=\overline{1,\ 5}}\mu^{(\mathrm{par})}(N,\ M;\ w) \tag{121}$$

are to be considered. Route gains (98) and (99) for parallelizing the TSPs generated by (88) — (91) are shown in Table 1 (where the gain values greater than 1 are highlighted bold). It is seen that the route gain with respect to the whole TSP grows as the number of nodes increases. As the depth of clustering is increased, route gain (98) deteriorates. This trend is seen in every row with (104) — (106). Route gain (99) seems to slightly deteriorate as well, but only TSPs with 1751 and 7001 nodes have this trend strictly confirmed. On average, the route gain with respect to the centroid-based approach is not that significant. In fact, both the serpentine-based and centroid-based approaches have the same accuracy on average. Nevertheless, speed gains (100) and (101) shown in Table 2 confirm that the serpentine-based approach is faster than the centroid-based approach, and it is far faster than solving the TSP without parallelization. As the depth of clustering is increased, the whole TSP speed gain grows. It slightly deteriorates as the number of nodes increases. The centroid-based TSP speed gain does not have such regularity, though.

Solving each of the five TSPs with 7001 nodes clustered in four clusters has taken the maximal number of iterations, as well as solving the whole TSP (i. e., 1.4 million iterations). This is why speed gain (100) is just 1 therein, and so are its values (110) — (112). Solving four of the five TSPs with 5251 nodes clustered in four clusters has taken also its maximal number of iterations (i. e., 1.05 million iterations for solving the whole TSP, and 1.05 million iterations in total for solving the four open-loop TSPs).

If the possibility of solving on M parallel processor cores is considered, speed gains (102) and (103) shown in Table 3 confirm the computational speed superiority of the serpentine-based approach. With respect to the whole TSP, the superiority grows as the clustering is made deeper, although it deteriorates as the number of nodes increases. Nevertheless, the serpentine-based approach with 64 clusters is expectedly at least 90 times faster for a few thousand nodes. In solving on M parallel processor cores, the serpentine-based approach is expectedly faster than the centroid-based approach. A computational outlier has been registered, though. Thus, a TSP with 1751 nodes has been solved by the serpentine-based approach with four clusters, where the open-loop TSPs have taken 38459, 47836, 32603, 75,913 iterations (see Fig. 5, where the depot is marked as filled square, the connecting nodes $\{k_m^{***}\}_{m=1}^{3}$ are marked as circles, and the four subroutes are aggregated clockwise). By the centroid-based approach, these TSPs have taken 42961, 44342, 27646, 50,748 iterations (Fig. 6). Even added the iterations taken to solve the centroid TSP (despite this supplementary TSP looks pretty trivial, there are 81 iterations, whichever the TSP size is), non-sequential speed gain (103) comes out to be

$$\frac{81+50748}{75913}\approx 0.6696.$$

It is the very TSP whose sequential speed gain

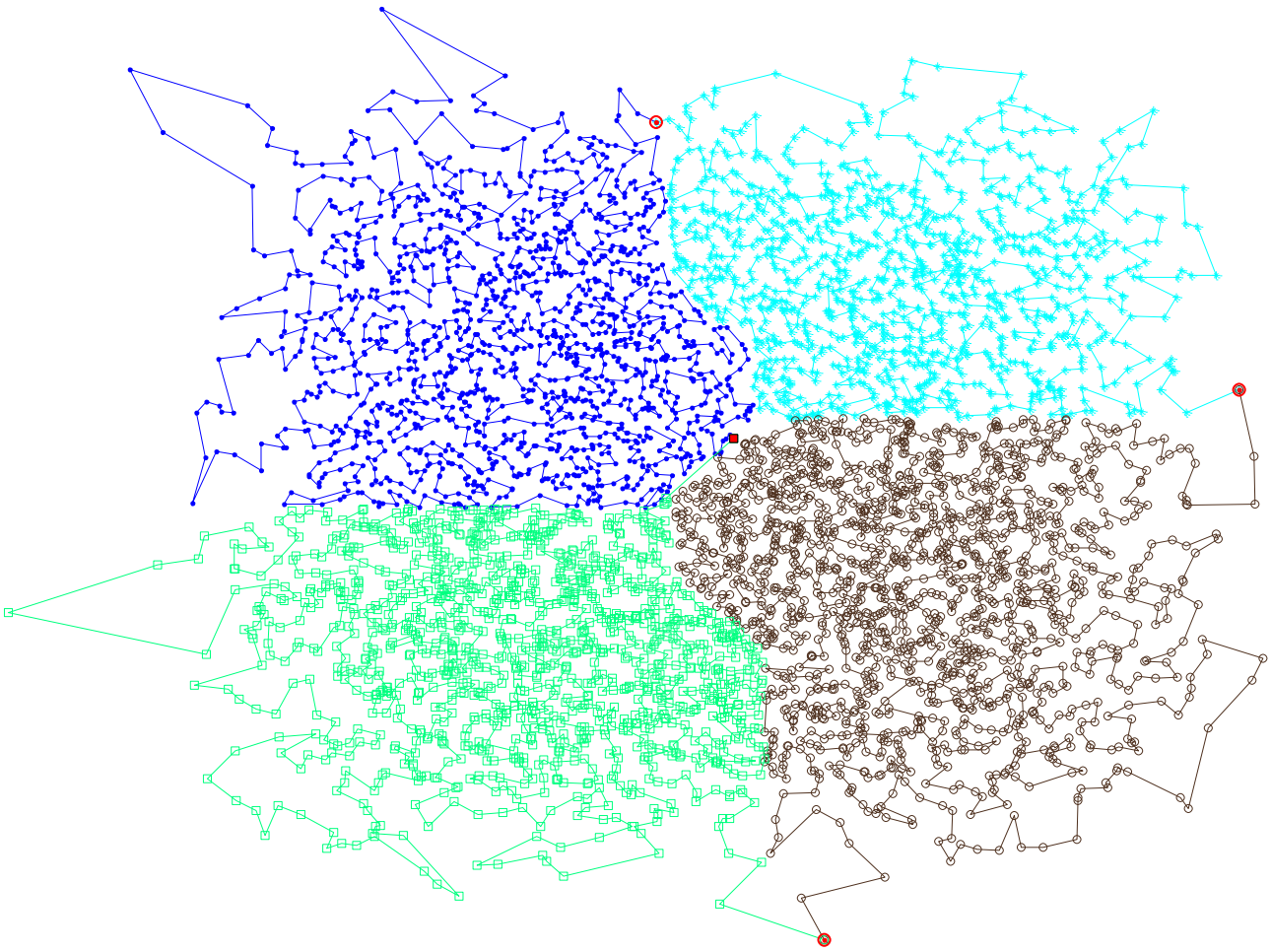


Fig. 7. The approximately shortest route aggregated clockwise from the four open-loop subroutes (whose respective lengths are 2085.3506, 1578.6051, 2158.6343, 1883.8827) by applying the serpentine-based approach of the parallelization.

$$\frac{81+42961+44342+27646+50748}{38459+47836+32603+75913}\approx 0.851$$

is minimal in Table 2. Despite the serpentine-based approach in this particular TSP has computed the solution 1.4934 times slower than the centroid-based approach, the latter is 1.0148 times less accurate. This is an example of the accuracy-versus-speed conflict that often happens. Therefore, an accuracy-versus-speed tradeoff should be always found.

Generally, Table 1 shows that the serpentine-based approach does not worsen the accuracy. However, solving every TSP without parallelization has taken the maximal number of iterations (which is 0.35, 0.7, 1.05, and 1.4 million iterations, respectively). So, maybe solving the whole TSP just needs more iterations to produce shorter routes than each of the serpentine-based and centroid-based approaches produces? In fact, it is likely to be true, and the likelihood grows as the TSP size increases. Indeed, as the number of nodes increases, the average number of iterations taken to solve the whole TSP grows faster than the sum of the respective averages taken to solve the open-loop subproblems.

An example of the TSP with 5251 nodes clustered into four groups is solved by having set the maximal number of iterations to $1000{\cdot}N_m$, $m=\overline{1,\ 4}$. There are 1420, 1076, 1538, and 1217 nodes in the respective four open-loop TSPs. The result of serpentine-based parallelization is presented in Fig. 7, where the open-loop TSPs have taken 527482, 323438, 676845, and 360,787 iterations, respectively (i. e., 29.65 % to 44.01 % of the iteration "capacity" has been used by each subproblem). The aggregated route length is 7706.4726, but the centroid-based parallelization has happened to be a bit more accurate (Fig. 8), where the aggregated route length is 7680.0877. Thus, the centroid-based approach in this example produces a 1.0034 times shorter route (it is 0.34 % shorter than the route in Fig. 7). Nevertheless, the open-loop TSPs have taken 527408, 331317, 679792, and 372,563 iterations (30.61 % to 44.2 % of the iteration "capacity" has been used by each subproblem), respectively, and the centroid TSP has taken 401 iterations. So, the sequential centroid-based parallelization has lasted

$$\frac{401+527408+331317+679792+372563}{527482+323438+676845+360787}\approx 1.0121$$

times longer. If all the four open-loop TSPs are solved in parallel (simultaneously on four processor cores), then the centroid-based parallelization appears to have been slower also:

$$\frac{401+\max\{527408,\ 331317,\ 679792,\ 372563\}}{\max\{527482,\ 323438,\ 676845,\ 360787\}}\approx 1.0049.$$

For solving the whole TSP in this example, the maximal number of iterations is set to $15.75{\cdot}10^6$. The approximately shortest route length is 7591.2768 (Fig. 9), which is 1.0117 times shorter than the aggregated route by centroid-based parallelization, and 1.0152 times shorter than the aggregated route by serpentine-based parallelization. The difference might be significant if the operational time and computational resources were not considered. It is a matter of not only the whole TSP has taken 4.3536 times more iterations (by the worst, sequential parallelization), but also of the whole TSP algorithm iteration that runs much slower.

A TSP with a half million nodes is an example of the large TSP. To deal with such an instance, a half million nodes (not counting the depot)

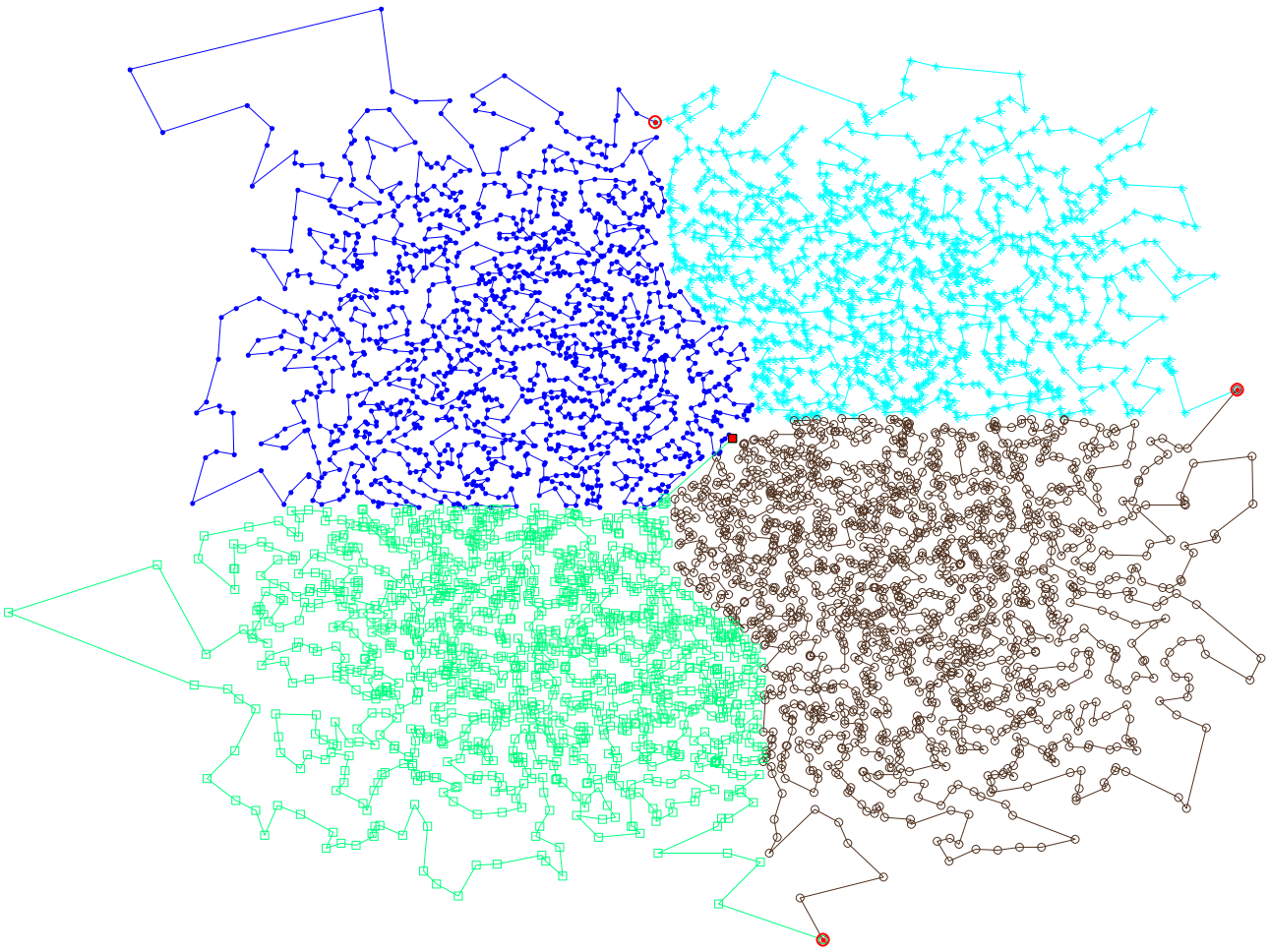


Fig. 8. The approximately shortest route aggregated clockwise from the four open-loop subroutes (whose respective lengths are 2073.6751, 1584.2432, 2153.4697, 1868.6998) by applying the centroid-based approach of the parallelization.

clustered into 1024 groups (i. e., it is the clustering of the 10 four steps depth) is solved by having set the maximal number of iterations to $1000{\cdot}N_m$, $m=\overline{1,\ 1024}$. The number of nodes in the respective 1024 open-loop TSPs varies between 91 and 1128, where the average number of nodes is 488.2813. The aggregated route length is 77724.1521 by the serpentine-based parallelization. The centroid-based parallelization has happened to be 0.975 % more accurate, where the aggregated route length is 76973.662. The difference is significant. None of the 2048 open-loop subroutes has self-intersections (Fig. 10). The serpentine-based open-loop TSPs have taken 10,381 (the very open-loop TSP with 91 nodes) to 390,820 iterations, where the average number of iterations is 98838.77 (which is 20.24 % of the iteration "capacity"). The same outputs (minimal, mean, and maximal values) for the centroid-based open-loop TSPs are 10486, 98995.1, and 382,160 iterations, respectively. Meanwhile, the centroid TSP has taken 271,775 iterations. Therefore, the sequential centroid-based parallelization has lasted

$$\frac{271775+\sum_{m=1}^{1024} l_{\Sigma}^{(\mathbf{C})*(m)}(N_m,\ 1024;\ 1)}{\sum_{m=1}^{1024} l_{\Sigma}^{\exists*(m)}(N_m,\ 1024;\ 1)}\approx 1.0043$$

times longer. In real-time units, the 1024 serpentine-based open-loop TSPs have been solved in 125.5468 h, whereas the sequential centroid-based parallelization has lasted for 126.6123 h (which is 1.0085 times longer) on CPU Intel Core i5-7200U@2.50 GHz. If all the 1024 open-loop TSPs are solved in parallel, then the centroid-based parallelization is significantly slower:

$$\frac{271775+\max\limits_{m=\overline{1,\ 1024}} l_{\Sigma}^{(\mathbf{C})*(m)}(N_m,\ 1024;\ 1)}{\max\limits_{m=\overline{1,\ 1024}} l_{\Sigma}^{\exists*(m)}(N_m,\ 1024;\ 1)}\approx 1.6732.$$

The initial TSP breaking with the deep clustering takes about 90 s for the half-million-nodes instance. Without the deep clustering, partitioning the nodes straightforwardly into 1024 groups lasts far longer consuming too much memory. For example, the half million nodes is partitioned into just 128 groups during almost 3 min, whereas 256 groups would be formed not earlier than in 7 min. By both the centroid-based and serpentine-based approaches, the aggregation of the subproblems solutions lasts about 20 ms for the half million nodes. The initial TSP breaking by the centroid-based approach, which essentially lies in solving the centroid TSP, takes about 23 min.

Other simulations of large and extremely large TSPs do not show a distinct advantage of either the centroid-based or serpentine-based approach. The simulations only confirm that the accuracy-versus-speed conflict exists in most cases of the sequential parallelization, where one approach is faster but less accurate and the other approach is slower but more accurate.

8. Discussion of the contribution

The deep clustering is an efficient method to approximately solve large-scale TSPs by quickly parallelizing their solution. The parallelization saves hours, days, and even months for large-scale TSPs not

Fig. 9. The approximately shortest route (without self-intersections) found in 8,221,930 iterations by solving the whole TSP.

worsening the route quality on average. Compared to straightforward clustering, the deep clustering is a significant improvement of the initial TSP breaking. It is also adjustable to the current maximal size of the open-loop TSP — the clustering progresses as deep as it is needed to solve the respective open-loop TSPs within reasonable amount of time and available memory.

Whether it is the centroid-based or serpentine-based approach, the deep clustering is used anyway to determine centroids (26). This is followed by determining the aggregation pattern that is either the serpentine by (44) — (49) or the centroid TSP solution. There is no more favorable pattern, though.

The parallelization is practically applicable to TSPs with a few hundred nodes and more. The number of clusters is reasonably selected such that the average cluster size be of up to a few hundred nodes. This so due to the genetic algorithm converges faster to nearly the best open-loop subroute when there are a few hundred nodes or less. Another merit of the parallelization is the possibility of solving each subproblem over multiple runs by setting the pseudorandom number generator to different seeds (Matsumoto and Nishimura, 1998; Kneusel, 2018). Then the best solution for each open-loop TSP is sent for aggregation into a close-loop route.

The significance of the suggested parallelization approach is hard to overestimate. It opens an elegant way to approximately solve any extremely large TSP. Moreover, both the centroid-based and serpentine-based approaches can be applied (including their multiple runs), whereupon the shorter aggregated route is selected. Their only limitations are the number of available processors and the size of the largest contiguous free memory block, which is equivalent to the maximum possible array. The latter, however, is successfully overcome by applying a simplified version of the clustering. In this case, the clustering pattern is a rectangular lattice itself, whereupon a set of nodes is divided into a lattice of subsets — clusters, each of which has nearly a square shape. For instance, when the simplified clustering is applied to the Mona Lisa TSP (see Fig. 11, where the depot is marked as filled square, and the connecting nodes $\{k_m^{***}\}_{m=1}^{63}$ are marked as circles), being the most famous large-scale TSP used for benchmarking (Honda et al., 2013), the resulting image reconstruction looks acceptable (Fig. 12).

In Fig. 11, the number of nodes in the cluster varies between 158 (intrinsic to sparser parts of nodes) and 3102 (intrinsic to denser parts). Solving all the subproblems on a single processor core has taken 53,388,914 iterations (it is 186.123 h versus 11.5 CPU years, as it is stated on math.uwaterloo.ca/tsp/data/ml/monalisa.html). The solution route length is 6210696.7489 (it is rounded) being 7.88 % longer than the lower bound (Honda et al., 2013). However, it is just 0.1231 % accuracy loss on average per open-loop TSP that is acceptable. Moreover, if all the open-loop TSPs for Fig. 11 are solved on 64 parallel cores, the computation lasts at most for 639 min (it is the computational time of the open-loop TSP with 3102 nodes).

The similar application of the 8×8 rectangular lattice to other well-known large TSP instances produces almost the same acceptable result. Table 4 shows results of deep clustering compared to six large-scale benchmark TSPs (their data are taken from math.uwaterloo.ca/tsp/data/index.html). The parallelization dramatically speeds up the solving process, while accuracy losses are far less significant. The accuracy-versus-speed tradeoff is quite acceptable here, especially if one primarily aims at quickly estimating the upper bound of the route.

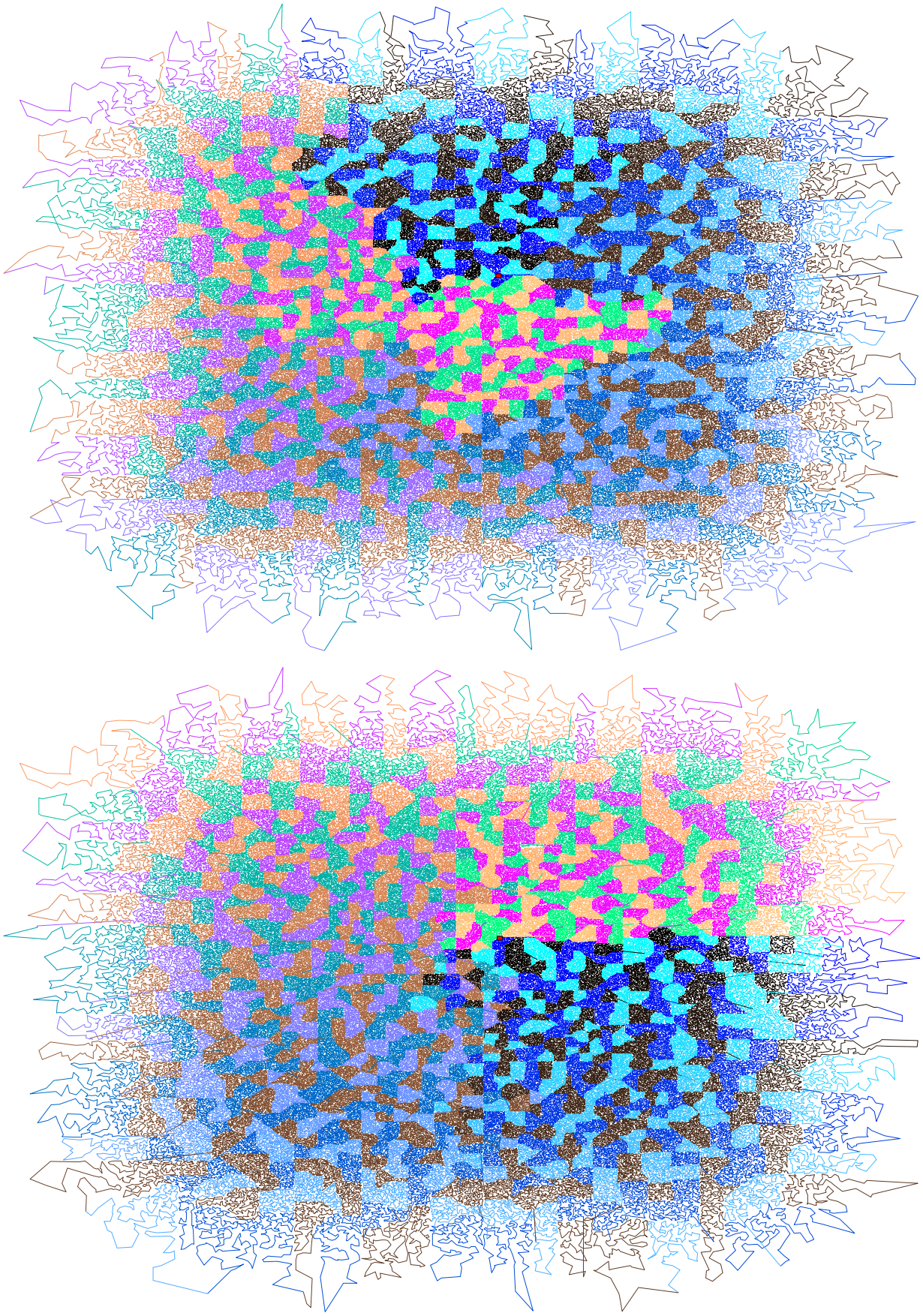


Fig. 10. The aggregated close-loop routes by the centroid-based (top) and serpentine-based (bottom) approaches as approximate solutions of the half-million-nodes TSP.

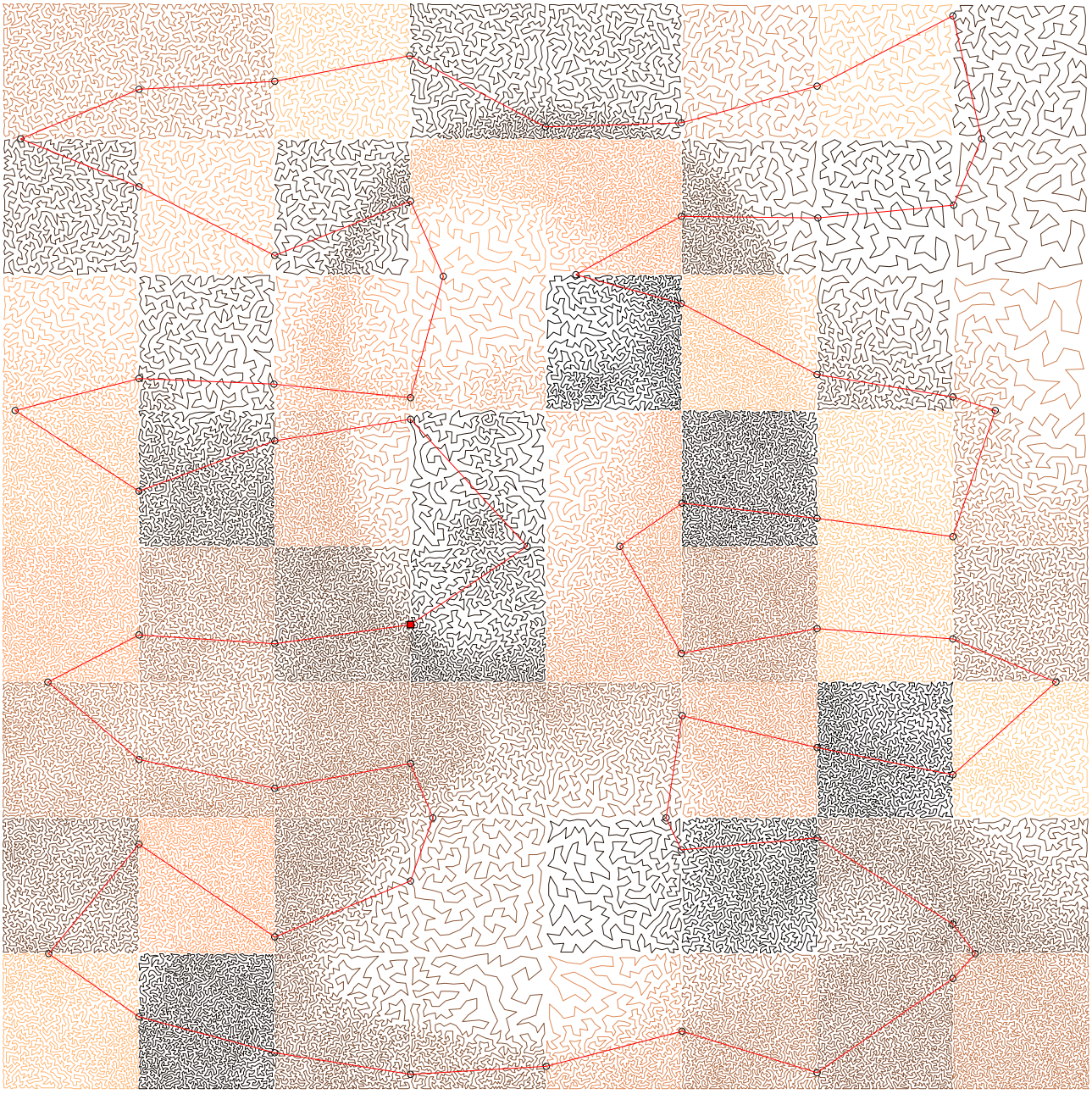


Fig. 11. The 8×8 rectangular lattice breaking the initial 0.1 million nodes of the Mona Lisa TSP into 64 clusters (highlighted by differing colors), whereupon the respective 64 open-loop TSPs are solved and then the close-loop route is aggregated by a symmetric rectangular close-loop serpentine; the connecting nodes $\{k_m^{***}\}_{m=1}^{63}$ are determined by (27) — (43).

It is hard to confidently estimate an expected gain by embedding the parallelization in genetic algorithms, but it is surely a boost for the non-sequential parallelization. Furthermore, extremely large TSPs with millions of nodes cannot be solved even approximately by the existing heuristics within reasonable amount of time, whereas both the centroid-based and serpentine-based approaches can tackle any intractably large TSP with up to a billion of nodes.

9. Conclusion

The deep clustering of the TSP breaks it into a set of much smaller open-loop TSPs, which are quickly solved in any order and then their solutions as open-loop subroutes are aggregated into the close-loop route as an approximate solution of the initial bigger-sized TSP. The initial TSP breaking is performed the fastest by the deep clustering, where the eventual number of clusters is a power of 2. The final number of clusters depends on the number of available processors and available memory with the maximum possible array. The aggregation pattern is far more determinative, and there are two nearly competitive approaches, which can be applied in parallel as well. By the centroid-based approach, the open-loop subroutes are aggregated by the shortest closed loop route passing through the centroids of the clusters. The aggregation route is found as the respective centroid TSP solution. By the serpentine-based approach, the open-loop subroutes are aggregated by a symmetric rectangular close-loop serpentine passing through the rectangular lattice cell centers approximately corresponding to the centroids. Two successive subroutes are connected so that the distance between the connecting nodes be approximately minimal. However, the starting and

Fig. 12. The Mona Lisa image reconstructed by the aggregated close-loop route in Fig. 11 (the clusters are discolored, and the aggregation route is removed).

Table 4
Deep clustering with the rectangular lattice on large-scale TSPs.

Large-scale TSP	Number of nodes	Known lower bound or best route length	Route length by deep clustering	Gap	Computational time on 64 cores	Known computational time
Mona Lisa	10^5	5,757,084	6210696.7489	7.88 %	639 min	11.5 CPU years
BBY34656	34,656	99,159	99327.5383	0.17 %	942 s	—
PBA38478	38,478	108,318	108523.1801	0.19 %	965 s	482,220 s
FNA52057	52,057	147,789	148989.224	0.81 %	1811 s	—
BNA56769	56,769	158,078	159629.096	0.98 %	1981 s	—
DAN59296	59,296	165,371	167109.233	1.05 %	3043 s	—

destination nodes of the open-loop TSP are distanced as farther as possible. In the very first open-loop TSP, the starting node is the depot and the destination node is maximally distanced from the depot by minimizing the distance to the second open-loop TSP cluster. In the other open-loop TSPs, except for the last one, the starting node is the destination node of the preceding open-loop TSP and the destination node is maximally distanced from it by minimizing the distance to the next open-loop TSP cluster. In the last open-loop TSP, the starting node is the destination node of the penultimate open-loop TSP and the destination node is the depot.

Therefore, the deep clustering, the initial TSP breaking and aggregation are contributed to the TSP solution theory and practice. Another contribution is a modification of the genetic algorithm for an open-loop TSP. The TSP parallelization is efficient at least in the dramatic computational speedup maintained for any number of nodes. The expected gain is counted in saving computational days, weeks, and months for a few hundred thousands to millions of nodes.

A further research can be based on developing a parallelization method for multiple TSP, where at least two salesmen exist to complete their tours as non-intersecting parts of the TSP route (Song et al., 2003; Kota and Jarmai, 2015). The presented TSP breaking cannot be directly overwritten or adapted for this case because some open-loop TSPs, having fewer nodes, may be solved with just a single salesman. As an alternative, additional constraints must be imposed on the breaking to prevent too small-sized open-loop TSPs.

CRediT authorship contribution statement

Vadim V. Romanuke: Writing – review & editing, Writing – original draft, Visualization, Validation, Supervision, Software, Resources, Project administration, Methodology, Investigation, Formal analysis, Data curation, Conceptualization.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

References

Applegate, D.L., Bixby, R., Chvátal, V., Cook, W.J., 2007. The Traveling Salesman Problem: A Computational Study. Princeton University Press, Princeton, NJ, USA.

Archetti, C., Peirano, L., Speranza, M.G., 2022. Optimization in multimodal freight transportation problems: A Survey. Eur. J. Oper. Res. 299 (1), 1–20. https://doi.org/10.1016/j.ejor.2021.07.031.

Bäck, T.H.W., Kononova, A.V., van Stein, B., Wang, H., Antonov, K.A., Kalkreuth, R.T., de Nobel, J., Vermetten, D., de Winter, R., Ye, F., 2023. Evolutionary algorithms for parameter optimization—thirty years later. Evol. Comput. 31 (2), 81–122. https://doi.org/10.1162/evco_a_00325.

Balma, A., Mrad, M., Ladhari, T., 2023. Tight lower bounds for the Traveling Salesman Problem with draft limits. Comput. Oper. Res. 154, 106196 https://doi.org/10.1016/j.cor.2023.106196.

Celebi, M.E., Kingravi, H.A., Vela, P.A., 2013. A comparative study of efficient initialization methods for the *k*-means clustering algorithm. Expert Syst. Appl. 40 (1), 200–210. https://doi.org/10.1016/j.eswa.2012.07.021.

Chambers, L.D., 2000. The Practical Handbook of Genetic Algorithms. Chapman and Hall/CRC, Boca Raton, FL, USA.

Chen, T.S., Tang, K., Chen, G.L., Yao, X., 2012. A large population size can be unhelpful in evolutionary algorithms. Theor. Comput. Sci. 436, 54–70. https://doi.org/10.1016/j.tcs.2011.02.016.

Colorni, A., Dorigo, M., Maffioli, F., Maniezzo, V., Righini, G., Trubian, M., 1996. Heuristics from nature for hard combinatorial optimization problems. Int. Trans. Oper. Res. 3 (1), 1–21. https://doi.org/10.1016/0969-6016(102)00004-4.

Crowder, H., Padberg, M.W., 1980. Solving large-scale symmetric traveling salesman problems to optimality. Manag. Sci. 26, 495–509. https://doi.org/10.1287/MNSC.26.5.495.

Dantzig, G.B., 1963. Linear Programming and Extensions. Princeton University Press, Princeton, NJ, USA.

Dasgupta, D., Michalewicz, Z., 2013. Evolutionary Algorithms in Engineering Applications. Springer Science & Business Media.

Davendra, D., and Zelinka, I. (2010). Controller Parameters Optimization on a Representative Set of Systems Using Deterministic-Chaotic-Mutation Evolutionary Algorithms. In I. Zelinka, S. Celikovsky, H. Richter, and G. Chen (Eds.), Evolutionary Algorithms and Chaotic Systems. Studies in Computational Intelligence, Vol. 267, pp. 447–480. Berlin, Heidelberg, Germany. 10.1007/978-3-642-10707-8_14.

Du, D.-Z., Pardalos, P.M., 1998. Handbook of Combinatorial Optimization. Springer, New York, NY, USA. https://doi.org/10.1007/978-1-4613-0303-9.

Fiechter, C.N., 1994. A parallel tabu search algorithm for large traveling salesman problems. Discret. Appl. Math. 51, 243–267. https://doi.org/10.1016/0166-218X(98)00033-I.

Fischetti, M., Lodi, A., Toth, P., 2002. Exact methods for the asymmetric traveling salesman problem. In: Gutin, G., Punnen, A.P. (Eds.), The Traveling Salesman Problem and Its Variations. Kluwer, Boston, MA, USA, pp. 169–205. https://doi.org/10.1007/0-306-48213-4_4.

Gonzalez, T., 1985. Clustering to minimize the maximum intercluster distance. Theor. Comput. Sci. 38, 293–306. https://doi.org/10.1016/0304-3975(85)90224-5.

Grötschel, M., Holland, O., 1991. Solution of large-scale symmetric traveling salesman problems. Math. Program. 51, 141–202. https://doi.org/10.1007/BF01586932.

Hansen, N., Ostermeier, A., 2001. Completely derandomized self-adaptation in evolution strategies. Evol. Comput. 9 (2), 159–195. https://doi.org/10.1162/106365601750190398.

Hartigan, J.A., Wong, M.A., 1979. Algorithm AS 136: A *k*-means clustering algorithm. J. R. Stat. Soc. Ser. C 28 (1), 100–108. https://doi.org/10.2307/2346830.

Haupt, R.L., Haupt, S.E., 2003. Practical Genetic Algorithms. John Wiley & Sons, Hoboken, NJ, USA. https://doi.org/10.1002/0471671746.

Hertz, A., Widmer, M., 2003. Guidelines for the use of meta-heuristics in combinatorial optimization. Eur. J. Oper. Res. 151 (2), 247–252. https://doi.org/10.1016/S0377-2217(02)00823-8.

Honda, K., Nagata, Y., Ono, I. (2013). A parallel genetic algorithm with edge assembly crossover for 100,000-city scale TSPs. In Proceedings of the 2013 IEEE Congress on Evolutionary Computation, pp. 1278–1285.

Ikotun, A.M., Ezugwu, A.E., Abualigah, L., Abuhaija, B., Heming, J., 2023. *K*-means clustering algorithms: A comprehensive review, variants analysis, and advances in the era of big data. Inf. Sci. 622, 178–210. https://doi.org/10.1016/j.ins.2022.11.139.

Kneusel, R., 2018. Random Numbers and Computers. Springer, Cham, Switzerland. https://doi.org/10.1007/978-3-319-77697-2.

Kota, L., Jarmai, K., 2015. Mathematical modeling of multiple tour multiple traveling salesman problem using evolutionary programming. App. Math. Model. 39 (12), 3410–3433. https://doi.org/10.1016/j.apm.2014.11.043.

Kramer, O., 2010. A review of constraint-handling techniques for evolution strategies. Applied Computational Intelligence and Soft Computing 2010 (3), 1–19. https://doi.org/10.1155/2010/185063.

Land, A., 2021. The solution of some 100-city travelling salesman problems. EURO Journal on Computational Optimization 9, 100017. https://doi.org/10.1016/j.ejco.2021.100017.

LaTorre, A., Peña, J. M., Robles, V., and Muelas, S. (2008). 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, pp. 1119–1120. 10.1145/1389095.1389307.

Lawler, E.L., Lenstra, J.K., Rinnooy Kan, A.H.G., Shmoys, D.B., 1985. The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization. Wiley, Chichester, UK.

Li, J., Li, M., 2013. An analysis on convergence and convergence rate estimate of elitist genetic algorithms in noisy environments. Optik 124 (24), 6780–6785. https://doi.org/10.1016/j.ijleo.2013.05.101.

Luo, Y., Golden, B., Poikonen, S., Zhang, R., 2022. A fresh look at the Traveling Salesman Problem with a Center. Comput. Oper. Res. 143, 105748 https://doi.org/10.1016/j.cor.2022.105748.

Manfrin, M., Birattari, M., Stützle, T., and Dorigo, M. (2006). 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, pp. 224–234. Lecture Notes in Computer Science, Vol. 4150.

Matsumoto, M., Nishimura, T., 1998. Mersenne twister: a 623-dimensionally equi-distributed uniform pseudo-random number generator. ACM Trans. Model. Comput. Simul. 8 (1), 3–30. https://doi.org/10.1145/272991.272995.

Mulder, S.A., Wunsch, D.C., 2003. Million city traveling salesman problem solution by divide and conquer clustering with adaptive resonance neural networks. Neural Netw. 16 (5), 827–832. https://doi.org/10.1016/S0893-6080(03)00130-8.

Nagata, Y., Kobayashi, S., 1999. The proposal and evaluation of a crossover for traveling salesman problems: edge assembly crossover. *Journal of the Japanese Society for*. Artif. Intell. 14 (5), 848–859. https://doi.org/10.11517/jjsai.14.5_848.

Orman, A.J., Williams, H.P., 2006. A Survey of Different Integer Programming Formulations of the Travelling Salesman Problem. In: Kontoghiorghes, E., Gatu, C. (Eds.), Optimisation, Econometric and Financial Analysis. Advances in Computational Management Science, Vol. 9. Springer, Berlin, Heidelberg, Germany, pp. 91–104.

Papadimitriou, C.H., Steiglitz, K., 1998. Combinatorial Optimization: Algorithms and Complexity. Dover Publications, Mineola, NY, USA.

Phillips, S. J. (2002). Acceleration of K-Means and Related Clustering Algorithms. In D. M. Mount and C. Stein (Eds.), Lecture Notes in Computer Science, Vol. 2409, pp. 166–177. Springer. 10.1007/3-540-45643-0_13.

Rego, C., Gamboa, D., Glover, F., Osterman, C., 2011. Traveling salesman problem heuristics: Leading methods, implementations and latest advances. Eur. J. Oper. Res. 211 (3), 427–441. https://doi.org/10.1016/j.ejor.2010.09.010.

Rocha, Y., Subramanian, A., 2023. Hybrid genetic search for the traveling salesman problem with hybrid electric vehicle and time windows. Comput. Oper. Res. 155, 106223 https://doi.org/10.1016/j.cor.2023.106223.

Rojas Cruz, J.A., Pereira, A.G.C., 2013. The elitist non-homogeneous genetic algorithm: Almost sure convergence. Statist. Probab. Lett. 83 (10), 2179–2185. https://doi.org/10.1016/j.spl.2013.05.025.

Schneider, J., Froschhammer, C., Morgenstern, I., Husslein, T., Singer, J.M., 1996. Searching for backbones — an efficient parallel algorithm for the traveling salesman problem. Comput. Phys. Commun. 96 (2–3), 173–188.

Shafiee, A., Arab, M., Lai, Z., Liu, Z., and Abbas, A. (2016). Automated process flowsheet synthesis for membrane processes using genetic algorithm: role of crossover operators. In Z. Kravanja and M. Bogataj (Eds.), Computer Aided Chemical Engineering, Vol. 38, pp. 1201–1206. Elsevier, Amsterdam, Netherlands. 10.1016/B978-0-444-63428-3.50205-8.

Song, C., Lee, K., and Lee, W. D. (2003). Extended simulated annealing for augmented TSP and multi-salesmen TSP. In Proceedings of the International Joint Conference on Neural Networks, Portland, OR, USA, pp. 2340–2343. 10.1109/IJCNN.2003.1223777.

Song, X. M., Li, B., and Yang, H. M. (2006). Improved Ant Colony Algorithm and its Applications in TSP. In Proceedings of Intelligent Systems Design and Applications, Jian, China, pp. 1145–1148. 10.1109/ISDA.2006.253773.

Tayarani-N, M.-H., Prügel-Bennett, A., 2016. An analysis of the fitness landscape of travelling salesman problem. Evol. Comput. 24 (2), 347–384. https://doi.org/10.1162/EVCO_a_00154.

Teixeira, E.D.S., de Araujo, S.A., 2024. Formulations for the clustered traveling salesman problem with *d*-relaxed priority rule. Comput. Oper. Res. 161, 106402 https://doi.org/10.1016/j.cor.2023.106402.

Tinós, R., Whitley, D., Ochoa, G., 2020. A new generalized partition crossover for the traveling salesman problem: tunneling between local optima. Evol. Comput. 28 (2), 255–288. https://doi.org/10.1162/evco_a_00254.

Toaza, B., Esztergár-Kiss, D., 2023. A review of metaheuristic algorithms for solving TSP-based scheduling optimization problems. Appl. Soft Comput. 148, 110908 https://doi.org/10.1016/j.asoc.2023.110908.

Toth, P., Vigo, D., 2002. Branch-and-bound algorithms for the capacitated VRP. In: Toth, P., Vigo, D. (Eds.), The Vehicle Routing Problem. SIAM Monographs on Discrete Mathematics and Applications. SIAM, Philadelphia, PA, USA, pp. 29–51.

Valenzuela, C.L., Jones, A.J., 1993. Evolutionary divide and conquer (I): A novel genetic approach to the TSP. Evol. Comput. 1 (4), 313–333. https://doi.org/10.1162/evco.1993.1.4.313.

Van Stein, B., Wang, H., Kowalczyk, W., Emmerich, M., Bäck, T., 2020. Cluster-based Kriging approximation algorithms for complexity reduction. Appl. Intell. 50 (3), 778–791. https://doi.org/10.1007/s10489-019-01549-7.

Wang, H., Emmerich, M., Bäck, T., 2014. Mirrored orthogonal sampling with pairwise selection in evolution strategies. In: Proceedings of the 29th Annual ACM Symposium on Applied Computing, pp. 154–156.

Woeginger, G. J. (2003). Exact Algorithms for NP-Hard Problems: A Survey. In M. Jünger, G. Reinelt, and G. Rinaldi (Eds.), Combinatorial Optimization — Eureka, You Shrink!. Lecture Notes in Computer Science, Vol. 2570, pp. 185–207. Springer, Berlin, Heidelberg, Germany. 10.1007/3-540-36478-1_17.

Zhang, P., Wang, J., Tian, Z., Sun, S., Li, J., Yang, J., 2022. A genetic algorithm with jumping gene and heuristic operators for traveling salesman problem. Appl. Soft Comput. 127, 109339 https://doi.org/10.1016/j.asoc.2022.109339.

Zheng, J., Zhong, J., Chen, M., He, K., 2023. A reinforced hybrid genetic algorithm for the traveling salesman problem. Comput. Oper. Res. 157, 106249 https://doi.org/10.1016/j.cor.2023.106249.

Zhong, Y., Lin, J., Wang, L., Zhang, H., 2018. Discrete comprehensive learning particle swarm optimization algorithm with Metropolis acceptance criterion for traveling salesman problem. Swarm Evol. Comput. 42, 77–88. https://doi.org/10.1016/j.swevo.2018.02.017.