Scientific Research Publishing

American Journal of Operations Research, 2018, 8, 133-166
http://www.scirp.org/journal/ajor
ISSN Online: 2160-8849
ISSN Print: 2160-8830

Solving the Traveling Salesman Problem Using Hydrological Cycle Algorithm

Ahmad Wedyan*, Jacqueline Whalley, Ajit Narayanan

School of Engineering, Computer and Mathematical Sciences, Auckland University of Technology, Auckland, New Zealand
Email: *ahmad.wedyan@aut.ac.nz, jacqueline.whalley@aut.ac.nz, ajit.narayanan@aut.ac.nz

How to cite this paper: Wedyan, A., Whalley, J. and Narayanan, A. (2018) Solving the Traveling Salesman Problem Using Hydrological Cycle Algorithm. *American Journal of Operations Research*, **8**, 133-166.
https://doi.org/10.4236/ajor.2018.83010

Received: February 22, 2018
Accepted: May 22, 2018
Published: May 25, 2018

Abstract

In this paper, a recently developed nature-inspired optimization algorithm called the hydrological cycle algorithm (HCA) is evaluated on the traveling salesman problem (TSP). The HCA is based on the continuous movement of water drops in the natural hydrological cycle. The HCA performance is tested on various geometric structures and standard benchmarks instances. The HCA has successfully solved TSPs and obtained the optimal solution for 20 of 24 benchmarked instances, and near-optimal for the rest. The obtained results illustrate the efficiency of using HCA for solving discrete domain optimization problems. The solution quality and number of iterations were compared with those of other metaheuristic algorithms. The comparisons demonstrate the effectiveness of the HCA.

Keywords

Water-Based Optimization Algorithms, Nature-Inspired Computing, Discrete Optimization Problems, NP-Hard Problems

1. Introduction

Nature provides inspiration that can be used for computational processes. Many nature-inspired algorithms have emerged for solving optimization problems. The HCA is one of the newly proposed algorithms in the field of the swarm intelligence. The HCA is a water-based algorithm that simulates water movement through the hydrological cycle. The HCA uses a collection of artificial *water drops* that pass through various hydrological water cycle stages in order to generate solutions. The algorithm has been divided into four main stages: *flow*, *evaporation*, *condensation*, and *precipitation*. Each stage has a counterpart in the natural hydrological cycle and has a role in constructing the solution. Moreover, these stages work to complement each other and occur sequentially. The result of one stage is input to the next stage. Temperature is the main factor driving the water cycle through all stages. The algorithm starts with a low temperature and gradually increases until the cycle begins, then the temperature drops, as is natural in the real hydrological cycle.

Water-based algorithms are considered to be a subclass of nature-inspired algorithms that are based on certain factors or processes related to the activities and natural movements of water. Therefore, they share certain aspects of their conceptual framework. Each algorithm has a set of parameters and operations that form a procedure used to find a solution in an iterative process. However, they differ in their mathematical models and stages. These algorithms are frequently and widely used in solving many optimization problems.

Although there are already several water-based algorithms, none of them takes into account the full water cycle and the activities associated with water movement. The partial simulation of a natural process may limit the algorithm performance, especially in terms of exploration and exploitation capabilities which can lead to problems such as stagnation, increased computational effort, or premature convergence. Adding extra stages to an algorithm should only be done when there are clear advantages in doing so. One of the aims of this paper is to provide evidence that, for solving the TSP, including all stages of the water cycle has benefits over including only some stages.

The intelligent water drops (IWD) algorithm is a water-based algorithm proposed by Shah-Hosseini [1]. The IWD was inspired by the natural flow behavior of water in a river and by what happens in the journey from water drops to the riverbed. The IWD algorithm has some weaknesses that affected its performance. The water drops update their velocity after they move from one place to another. However, this increase in the velocity is very small (imperceptible) and affects the searching capability. The update also does not consider that water drop velocity might also decrease. Soil can be only removed (no deposition mechanism), and that may lead to premature convergence or being trapped in local optima. In IWD only indirect communication is considered as represented by soil erosion. Finally, the IWD does not use evaporation, condensation, or precipitation. These additional stages can improve the performance of water drop algorithms and play an important role in the construction of better solutions.

The water cycle algorithm (WCA) is another water-based algorithm, proposed by Eskandar *et al.* [2]. The WCA is based on flow of river and stream water towards the sea. In WCA, entities are represented by a set of streams. These streams keep moving from one point to another, which simulates the flow process of the water cycle. The evaporation process occurs when the positions of the streams/rivers are very close to that of the sea. The WCA omits some important factors in the natural water cycle. In WCA, no consideration is made for soil removal from the paths, which is considered a critical operation in the formation of streams and rivers. There is no consideration also for the condensation stage in WCA, which is one of the crucial stages in the water cycle. On the other hand, the WCA and PSO algorithms share similar structures but use different nomenclatures for their components.

A major problem in some of these algorithms is the process of choosing the next point to visit. They use one heuristic for controlling the movement of the entities in the search space. In particular, this can be observed when most algorithm entities keep choosing the same nodes repeatedly because there is no other factor affecting their decisions. For instance, the IWD algorithm uses only the soil as heuristic for guiding the entities through the search space. For this reason, the IWD suffers from inability to make a different selection among a set of nodes that have similar probabilities [3]. One of the more common ways to address this problem is to include another heuristic that can affect the calculation of the probabilities. In HCA, the probability of selecting the next node is an association between two natural factors: the soil and the depth of the path, which enables the construction of a variety of solutions.

Furthermore, some existing particle swarm algorithms rely on either direct or indirect communication for sharing information among the entities. Enabling both direct and indirect communication leads to better results and may reduce the iterations to reach the global optimum solution. Otherwise, the entities are likely to fall into the local optimum solution or produce the same solutions in each iteration (stagnation), and this leads to a degradation of the overall performance of the algorithm.

These aspects have been considered when designing the HCA by taking into account the limitations and weaknesses of previous water-based algorithms. This refinement involved enabling direct and indirect communication among the water drops. Such information sharing improved the overall performance and solution quality of the algorithm. Indirect communication was achieved in the flow stage by depositing and removing soil on/from paths and using path-depth heuristics. Direct communication was implemented via the condensation stage and was shown to promote the exploitation of good solutions. Furthermore, the condensation is a problem-dependent stage that can be customized according to the problem specifications and constraints. The cyclic nature of the HCA also provided a self-organizing and a feedback mechanism that enhanced the overall performance. The search capability of the HCA was enhanced by including the depth factor, velocity fluctuation, soil removal and deposition processes. The HCA provides a better balance between exploration and exploitation processes by considering these features. This confirmed that changing certain design aspects can significantly improve the algorithm's performance. The HCA was successfully applied and evaluated on continuous optimization problems [4].

This paper aims to present a new approach for solving TSP using HCA. This application also helps to evaluate the performance of the HCA on a discrete domain problem. Although many approaches can solve the TSP with high quality, the TSP remains an effective way of testing a new algorithm on discrete problems. Therefore, the main goal of this application is to measure the algorithm's ability to optimize (or nearly optimize) the solution for a simple discrete

NP-hard problem. Through the success of this application, we can define the strength of HCA and whether it is able to deal with other NP-hard problems.

The rest of this paper is organized as follows. Section 2 provides an overview of some algorithms have been used to solve TSPs. Section 3 reviews the TSP and its formulation. Section 4 presents the configuration of HCA and explains its application to the TSP. Section 5 demonstrates the feasibility of solving TSP instances by HCA and compares the results with those of other algorithms. Discussion and conclusions are presented in Section 6.

2. Literature Review

In general, small TSPs are most easily solved by trying all possibilities (*i.e.* exhaustive searching). This can be achieved by brute-force and branch-and-bound. These methods generate all possibilities and choose the least-cost solution at various choice points. Although these techniques will guarantee the optimal solution, they become impractical and expensive (*i.e.* require unreasonable time) when solving large TSP instances. A simple alternative is a greedy heuristic algorithm, which solves the TSP using a heuristic function. Such algorithms cannot guarantee the optimal solution, as they do not perform an exhaustive search. However, they perform sufficiently many evaluations to find the optimal/near optimal solution. Many greedy algorithms have been developed for TSPs, such as the nearest-neighbor (NN), insertion heuristics, and dynamic programming (DP) techniques. Metaheuristic algorithms can also provide high-quality solutions to large TSP instances.

The TSP has been extensively solved by different metaheuristic algorithms owing to its practical applications. The IWD algorithm was tested on the TSP [1]. Experiments confirmed that the IWD algorithm can solve this problem and obtains good results in some instances. Later, Msallam and Hamdan [5] presented an improved adaptive IWD algorithm. The adaptive part changes the initial value of the soil and the velocity of the water drops during the execution. The change is made when the quality of the results no longer improves, or after a certain number of iterations. Moreover, the initial-value change was based on the obtained fitness value of each water drop. Msallam and Hamdan used some of the modifications proposed in Shah-Hosseini [6]; that is, the amount of soil along each edge is reinitialized to a common value after a specified number of iteration, except for the edges that belong to the best solution, which lose less soil. These modifications diversify the exploration of the solution space and help the algorithm to escape from local optima. When tested on the TSP, the new adaptive IWD algorithm outperformed the original IWD.

Wu, Liao, and Wang [7] tested the water wave optimization (WWO) algorithm on the TSP. In WWO, each wave generates a solution and its fitness is measured by the total cost of the tour. For the TSP, the WWO operators were adapted to handle problems with a discrete domain. The propagation operator mutated the tours with a probability equal to the wavelength. Therefore, a bad solution (*i.e.*, a long-wavelength solution) was more likely to be mutated. The refraction operator enhanced the tour by choosing a random subsequence of cities from the best solution found so far and replacing it with a subsequence of the original tour. The breaking operation generated a number of new waves by performing swap operations between two previous waves. The algorithm was tested on seven benchmark instances of different sizes. In comparison studies, the WWO algorithm competed well against the genetic algorithm and other optimization algorithms, and solved the TSP with good results, but with slightly longer computational time than the genetic algorithm.

The water flow-like algorithm (WFA) is also used to solve the TSP [8]. Initially, a set of solutions to the water-flow is generated using a nearest-neighbor heuristic. In successive iterations, they are moved by insertions and 2-Opt procedures. The evaporation and precipitation operations are unchanged from the original WFA. These processes repeat until the stopping criteria are met.

In solving the TSP using river formation dynamics (RFD), Rabanal, Rodríguez, and Rubio [9] represented the problem as a landscape with all cities initially at the same altitude. They adjusted the representation by cloning the start-point city, allowing water to return to that city. Water movement is affected by the altitude differences among the cities and the path distances. The solutions (tours) are represented as sequences of cities sorted by decreasing altitude. To prevent the water drops from immediately eroding the landscape after each movement, the algorithm is modified to erode all cities when the drop reaches the destination city. This modification prevents quick reinforcement and avoids premature convergence. When tested on a number of TSP instances, the algorithm obtained a better solution than ant colony optimization, but required a longer computational time. The authors concluded that the RFD algorithm is a good choice if the solution quality is more important than the computational time.

Zhan, Lin, Zhang, and Zhong [10] solved the TSP by simulated annealing (SA) and a list-based technique. The main objective was to simplify the tuning of the temperature value. The list-based technique stores a priority queue of values that control the temperature decrease. In each iteration, the list is adapted based on the solution search space. The maximum value in the list is assigned the highest probability of becoming a candidate temperature. The SA employs local-neighbor search operators such as 2-Opt, 3-Opt, insert, inverse, and swap. The effectiveness of this algorithm has been measured in variously sized benchmark instances. The obtained results were competitive with those of other algorithms.

Geng, Chen, Yang, Shi, and Zhao [11] solved the TSP by adaptive SA combined with a greedy search. The greedy search was intended to improve the convergence rate. The SA implemented three types of mutations with different probabilities: vertex insertion, block insertion, and block reversion. The algorithm was tested on sixty benchmark instances. The computational results confirmed the higher effectiveness of the SA algorithm (in terms of CPU time and accuracy) than other algorithms.

Genetic algorithm (GA) has also been applied to TSPs in different configurations [12] [13]. Larranaga, Kuijpers, Murga, Inza, and Dizdarevic [14] reviewed the different representations and operators of GAs in TSP applications. Other papers have surveyed the application of different GA versions to the TSP [15] [16] [17] [18].

Ant colony optimization (ACO) has been applied to the TSP ([19] [20]) on symmetric and asymmetric graphs [21]. For solving TSPs, Hlaing and Khine [22] initialized the ant locations by a distribution approach that avoids search stagnation, and places each ant at one city. The ACO is improved by a local optimization heuristic that chooses the next-closest city and by an information entropy that adjusts the parameters. When tested on a number of benchmark instances, the improved ACO delivered promising results; especially, the improvements increased the convergence rate over the original ACO.

Zhong, Zhang, and Chen [23] developed a modified discrete particle swarm optimization (PSO), called C3DPSO, for TSPs. C3 refers to a mutation factor that balances the exploitation and exploration in the update equation, buffers the algorithm against being trapped in local optima, and avoids premature convergence. The solution of each particle is represented as a set of consecutive edges, requiring modifications in the update equations. The C3DPSO was tested on six benchmark instances with fewer than 100 cities. The proposed algorithm yielded more precise solutions within less computational time than the original PSO algorithm. In [24], a new concept based on mobile operators and its sequence is used to update the positions of particles in PSO, and it has been tested on TSP.

Wang, Huang, Zhou, and Pang [25] solved the TSP by a PSO with various types of swap operations, which assist the algorithm in finding the best solutions. The swap operation exchanges the positions of two cities, or the sequence of cities between two routes. When tested on a 14-node problem, the algorithm searched only a small part of the search space due to its high convergence rate. In the TSP solution of Shi, Liang, Lee, Lu, and Wang [26], an uncertain searching technique is associated with the particle movements in PSO. The convergence speed is increased by a crossover operation that eliminates intersections in the tours. The update equations of the original PSO are modified to suit the TSP problem. The proposed algorithm was extended to TSPs by employing a generalized chromosome. On various benchmark instances, the proposed algorithm proved more efficient than other algorithms.

Other algorithms like the bat algorithm has also used to solve several TSPs [27] [28]. A review of Tabu Search applications on the TSP and its variations can be found in [29].

3. Problem Formulation

The TSP is a well-known classical combinatorial optimization problem in which a salesperson must visit every designated city exactly once, and return to the starting point, via the shortest possible route. Such a path is known as a Hamiltonian cycle [30]. For centuries, the TSP has attracted researchers' attention owing to the simplicity of its formulation and constraints. However, despite being easy to describe and understand, the TSP is difficult to solve [31]. Because a vast amount of information has been amassed on the TSP and the behaviors of TSP algorithms are easily observed, the TSP is now recognized as a standard benchmarking problem for evaluating new algorithms and comparing their performances with those of established algorithms. Many real-life problems and applications can also be formulated as TSPs, and some optimization problems with different structures can be reduced or transformed to variations of TSPs, such as the job scheduling problem, the knapsack problem, DNA sequencing, integrated circuit (*i.e.*, VLSI circuits) design, drilling problem, and the satisfiability problem. Finally, a TSP can be classified as a combinatorial optimization problem, as it requires finding the best solution from a finite set of feasible solutions.

Typically, a TSP is represented as a complete undirected weighted graph, where each node is connected to all other nodes. The graph $G = (V, E)$ consists of a set of V nodes (*i.e.* cities) connected by a set of E edges (*i.e.* roads), where the edges are associated (assigned) with various weights. The weight is a non-negative number reflecting the distance, the travel cost, or time of traveling that edge. Given the node coordinates (locations), the Euclidean distance between two nodes i and j can be calculated as follows:

$$Distance(i,j) = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} \tag{1}$$

The TSP can be a symmetric or asymmetric weighted problem. In the symmetric problem, the path from node A to node B has the same weight as the path from node B to node A. In contrast, paths in the asymmetric problem may be unidirectional or carry different weights in each direction. Mathematically, the TSP can be formulated as Equation (2) [31], where D_{ij} represents the distance between nodes i and j.

$$\text{Minimise} \sum_{i=1}^{N} D_{ij} X_{ij}, N \geq 3 \tag{2}$$

subject to

$$X_{ij} \in \{0,1\}, i, j = 1, \cdots, N, i \neq j \tag{3}$$

In Equation (3), the decision variables X_{ij} are set to 1 if the connecting edge is part of the solution, and 0 otherwise:

$$X_{ij} = \begin{cases} 1, & \text{if } (i,j) \in \text{Solution} \\ 0, & \text{if } (i,j) \notin \text{Solution} \end{cases} \tag{4}$$

The TSP is considered as an NP-hard problem, meaning that its complexity increases non-linearly with increasing number of cities. Therefore, the number of possible solutions rises rapidly as the number of cities increases. Practically, the TSP finds the best order of the visited nodes at the lowest cost, which can be interpreted as a permutation problem. The number of possible solutions for an *n*-city problem is given by:

$$\text{Number of solutions} = \frac{(n-1)!}{2}, \quad \text{where } n \geq 3 \tag{5}$$

Equation (5) calculates the number of possible ways of arranging *n* cities into an ordered sequence (with no repeats). As the starting node is unimportant, there are (*n* − 1)! rather than *n*! possible solutions. The result is divided by two because the reverse routes are ignored. **Figure 1** shows a simple TSP with five nodes.

In this example, one of the best solutions is (2 → 1 → 5 → 4 → 3 → 2) with a cost of 190. Another repeated solution with the same cost but a different starting node is (1 → 5 → 4 → 3 → 2 → 1).

4. The HCA-TSP Approach and Procedure

Typically, the input of the HCA algorithm is represented as a graph. To solve the TSP, the input to the HCA will be a fully connected graph that represents the problem solution space. The graph has a set of nodes (cities) and set of undirected edges (roads) between the nodes. The characteristics associated with each edge are the initial amount of soil and edge depth. The HCA uses a set of artificial water drops to generate solutions, where each water drop has three properties: velocity, amount of carried soil, and solution quality. The procedure of HCA is specified in the following steps:

1) Initialization of the variables and read the problem data.

2) Distribution of the water drops on the nodes of the graph at random.

3) Repeat steps 4) to 7) until termination conditions are met.

4) The flow stage (repeat sub-steps a) - d) until temperature reaches a specific value).

A water drop iteratively constructs a solution for the problem by continuously moving between the nodes.

a) Choosing next node

The movements are affected by the amount of soil and the path depths. The probability of choosing node *j* from node *i* is calculated using Equation (6).

$$P_i^{WD}(j) = \frac{f\left(Soil(i,j)\right)^2 \times g\left(Depth(i,j)\right)}{\sum_{k \notin vc(WD)} \left(f\left(Soil(i,k)\right)^2 \times g\left(Depth(i,k)\right) \right)} \tag{6}$$

where $P_i^{WD}(j)$ is the probability of choosing node *j* from node *i*, and *vc* is the

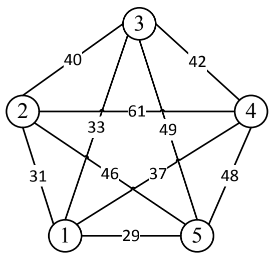


Figure 1. TSP instance with five nodes.

visited list of each water drop. The $f(Soil(i, j))$ is equal to the inverse of the soil between i and j, and is calculated using Equation (7).

$$f\left(Soil\left(i,j\right)\right)=\frac{1}{\varepsilon+Soil\left(i,j\right)} \tag{7}$$

$\varepsilon = 0.01$ is a small value that is used to prevent division by zero. The second factor of the transition rule is the inverse of depth, which is calculated based on Equation (8).

$$g\left(Depth\left(i,j\right)\right)=\frac{1}{Depth\left(i,j\right)} \tag{8}$$

Depth (i, j) is the depth between two nodes i and j, and calculated by dividing the length of the path by the amount of soil. The depth of the path needs to be updated when the amount of soil existing on the path changes. The depth is updated as follows:

$$Depth\left(i,j\right)=\frac{Length\left(i,j\right)}{Soil\left(i,j\right)} \tag{9}$$

After selecting the next node, the water drop moves to the selected node and marks it as visited.

b) Update velocity

The velocity of a water drop might be increased or decreased while it is moving. Mathematically, the velocity of a water drop at time (t + 1) is calculated using Equation (10).

$$V_{t+1}^{WD}=\left[K\times V_t^{WD}\right]+\alpha\left(\frac{V_t^{WD}}{Soil\left(i,j\right)}\right)+\sqrt[2]{\frac{V_t^{WD}}{Soil^{WD}}}+\left(\frac{100}{\psi^{WD}}\right)+\sqrt[2]{\frac{V_t^{WD}}{Depth\left(i,j\right)}} \tag{10}$$

where V_{t+1}^{WD} is the current water drop velocity, and K is a uniformly distributed random number between [0, 1] that refers to the roughness coefficient. Alpha (α) is a relative influence coefficient that emphasizes this term in the velocity update equation and helps the water drops to emphasize and favor the path with fewer soils over the other factors. The expression is designed to prevent one water drop from dominating the other drops. That is, a high-velocity water drop is able to remove more soil than slower ones. Consequently, the water drops are more likely to follow the carved paths, which may guide the swarm towards local optimal solution.

c) Update soil

Next, the amount of soil existing on the path and the depth of that path are updated. A water drop can remove (or add) soil from (or to) a path while moving based on its velocity. This is expressed by Equation (11).

$$Soil\left(i,j\right)=\begin{cases}\left[PN*Soil\left(i,j\right)\right]-\Delta Soil\left(i,j\right)-\sqrt[2]{\dfrac{1}{Depth\left(i,j\right)}} & \text{if } V^{WD}\geq Avg\left(all_V^{WDS}\right)\left(\text{Erosion}\right)\\ \left[PN*Soil\left(i,j\right)\right]+\Delta Soil\left(i,j\right)+\sqrt[2]{\dfrac{1}{Depth\left(i,j\right)}} & \text{else}\left(\text{Deposition}\right)\end{cases} \tag{11}$$

PN represents a coefficient (*i.e.*, sediment transport rate, or gradation coefficient) that may affect the reduction in the amount of soil. The increasing soil amount on some paths favors the exploration of other paths during the search process and avoids entrapment in local optimal solutions. The rate of change in the amount of soil existing between node *i* and node *j* depends on the time needed to cross that path, which is calculated using Equation (12).

$$\Delta Soil(i,j) = \frac{1}{time_{i,j}^{WD}} \tag{12}$$

such that,

$$time_{i,j}^{WD} = \frac{Distance(i,j)}{V_{t+1}^{WD}} \tag{13}$$

In HCA, the amount of soil the water drop carries reflects its solution quality. Therefore, the water drop with a better solution will carry more soil, which can be expressed by Equation (14).

$$Soil^{WD} = Soil^{WD} + \frac{\Delta Soil(i,j)}{\psi^{WD}} \tag{14}$$

One iteration is considered complete when all water drops have generated solutions based on the problem constraints (*i.e.*, when each water drop has visited each node). A solution represents the order of visiting all the nodes and returning to the starting node. The qualities of the evaluated solutions are used to update the temperature.

d) Update temperature

The new temperature value depends on the solution quality generated by the water drops in the previous iterations. The temperature will be increased as follows:

$$Temp(t+1) = Temp(t) + \Delta Temp \tag{15}$$

where,

$$\Delta Temp = \begin{cases} \beta * \left(\dfrac{Temp(t)}{\Delta D}\right) & \Delta D > 0 \\ \dfrac{Temp(t)}{10} & \text{otherwise} \end{cases} \tag{16}$$

and where coefficient β is determined based on the problem. The difference (ΔD) is calculated using Equation (17).

$$\Delta D = MaxValue - MinValue \tag{17}$$

Such that,

$$\begin{aligned} MaxValue &= \max\left[\text{Solutions Quality}(WDs)\right] \\ MinValue &= \min\left[\text{Solutions Quality}(WDs)\right] \end{aligned} \tag{18}$$

According to Equation (17), increase in temperature will be affected by the difference between the best solution (*MinValue*) and the worst solution (*MaxValue*). At the end of each iteration, the HCA checks whether the temperature is high enough to evaporate the water drops. Thus, the flow stage may run several times before the evaporation stage starts. When the temperature increases and reaches a specified value, the evaporation stage is invoked.

5) The evaporation stage:

A certain number of water drops evaporates based on the evaporation rate. The evaporation rate is determined by generating a random number between one and the total number of water drops (see Equation 19).

$$\text{Evaporation rate} = \text{Random\_Integer}(1, N) \tag{19}$$

The evaporated water drops are selected by the roulette wheel technique. The evaporation process is an approach to avoid stagnation or local-optimal solutions.

6) The condensation stage:

The condensation stage is executed as a result of the evaporation process, which is a problem-dependent process and can be customized to improve the solution quality by performing certain tasks (*i.e.*, local improvement method). The condensation stage collides and merges the evaporated water drops, eliminating the weak drops and favoring the best drop (*i.e.*, the collector), see Equation (20).

$$OP(WD_1, WD_2) = \begin{cases} \text{Bounce}(WD_1, WD_2), & \text{Similarity} < 50\% \\ \text{Merge}(WD_1, WD_2), & \text{Similarity} \geq 50\% \end{cases} \tag{20}$$

Finding the similarity between the solutions is problem-dependent, and measures how much two solutions are close to each other. For the TSP, the similarities between the solutions of the water drops are measured by the Hamming distance [32]. When two water drops collide and merge, one water drop will (*i.e.*, the collector) become more powerful by eliminating the other one and acquires its characteristics (*i.e.*, its velocity). The merging operation is useful to eliminate one of the water drops as they have similar solutions. On the other hand, when two water drops collide and bounce off, they will directly share information with each other about the goodness of each node, and how much a node contributes to their solutions. The bounce-off operation generates information that is used later to refine the water drops' solution quality in the next cycle by emphasis on the best nodes. The information is available and accessible to all water drops and helps them to choose a node that has a better contribution from all the possible nodes at the flow stage. For the TSP, the evaporated water drops share their information regarding the most promising nodes sequence. Within this exchange, the water drops will favor those nodes in the next cycle. Finally, the condensation stage is used to update the global-best solution found up to that point. With regard to temperature, determining the appropriate temperature values is through trial and error, and appropriate values for this problem were identified through experimentation. The values (**Table 1**) have been determined after some preliminary experiments with the TSP problem. The lowering and rising of the temperature not only control the cycle but also help to prevent the water drops from sticking with the same solution every iteration.

Table 1. HCA parameters and their values.

Parameter name	Parameter value
Number of water drops	Equal to number of nodes
Maximum number of iterations	Triple the number of nodes
Initial soil on each edge	10,000
Initial velocity	100
Initial depth	Edge length/soil on that edge
Initial carrying soil	1
Velocity updating	$\alpha = 2$
Soil updating	$PN = 0.99$
Initial temperature	50, $\beta = 10$
Maximum temperature	100

7) The precipitation stage:

This precipitation is considered as a termination stage, as the algorithm has to check whether the termination condition is met. If the condition has been met, the algorithm stops with the last global-best solution. Otherwise, this stage is responsible for reinitializing all the dynamic variables, such as the amount of the soil on each edge, depth of paths, the velocity of each water drop, and the amount of soil it holds. The re-initialization of the parameters happens after certain iterations and helps the algorithm to avoid being trapped in local optima, which may affect the algorithm's performance in the next cycle. Moreover, this stage is considered as a reinforcement stage, which is used to place emphasis on the collector drop. This is achieved by reducing the amount of soil on the edges that belong to the best water drop solution, see Equation (21).

$$Soil(i,j) = 0.9 * soil(i,j), \forall (i,j) \in Best^{WD} \tag{21}$$

The idea behind that is to favor these edges over the other edges in the next cycle. These stages are repeated until the maximum number of iterations is reached. The HCA goes through a number of cycles and iterations to find a solution to a problem. Figure 2 explains the steps in solving the TSP by HCA.

4.1. Solution Representation

In this paper, the TSP is assumed to be symmetric, and acting on a fully connected graph. The candidate TSP solutions are stored in a matrix, where each row represents a different solution generated by a water drop. Therefore, a water drop solution consists of the order of the visited nodes (with no repeat visits). The length of each row (*i.e.* the number of columns) is denoted by n and determined by the total number of nodes (see Equation 22).

$$\text{Solutions} = \begin{bmatrix} WD_1 & 1 & 2 & \cdots & n \\ WD_2 & 1 & 2 & \cdots & n \\ WD_3 & 1 & 2 & \cdots & n \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ WD_n & 1 & 2 & \cdots & n \end{bmatrix} \tag{22}$$

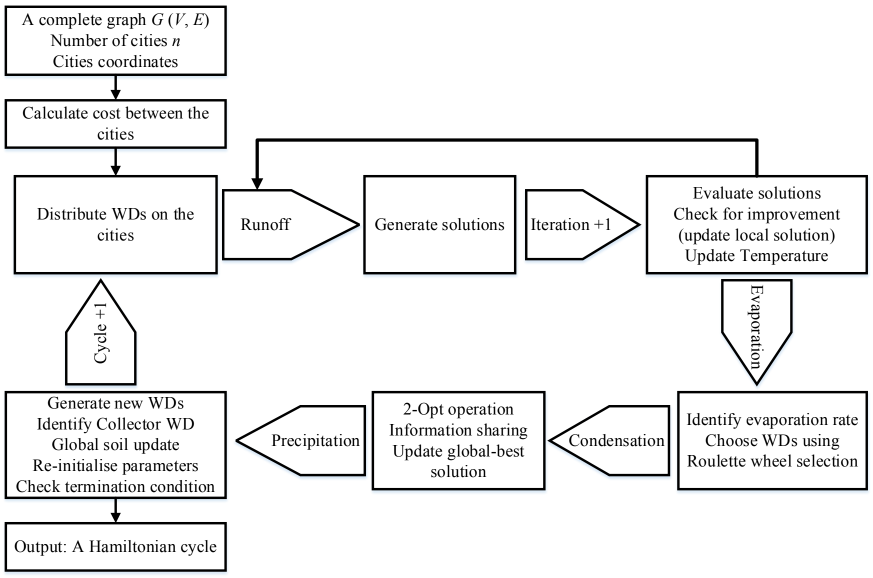


Figure 2. TSP solution procedure of HCA.

4.2. Local Improvement Operation

The quality of generated tours can be improved by many operations, such as k-Opt (where k = 2, 3, or 4) [33] [34]. These operations enhance the performance of the algorithm and minimize the number of iterations to reach the optimal solution. In the present problem, we apply the 2-Opt operation on the selected water drops that will evaporate at the condensation stage. The 2-Opt operation swaps the order of two edges at one part of the tour and keeps the tour connected. The swapping results in a new tour, which is accepted if it minimizes the total cost [35]. This operation is repeated until a stopping criterion is met, such as no further improvements after a certain number of exchanges, or when the maximum number of exchanges is reached. Figure 3 demonstrates the operation of 2-Opt. In this example, the algorithm selects edges (2, 7) and (3, 8), and consecutively creates new edges (2, 3) and (7, 8). The order of the nodes between the two edges must also be reversed.

5. Experimental Results and Analysis

The HCA was tested and evaluated on two groups of TSP instances; structural and benchmark. The runtime and solution quality of the benchmark results were compared with those of other algorithms.

The HCA parameter values used for TSP are listed in Table 1. The parameters values are set after conducting some preliminary experiments.

The depth values had a very small value. Therefore, it has been normalized to be within [1 - 100]. The amount of soil has been restricted to be within a maximum and minimum value for avoiding negative values. The maximum value is regarded as the initial value, while the minimum value is fixed to equal one. The algorithm was implemented using MATLAB. All the experiments were conducted on a computer with Intel Core i5-4570 (3.20 GHz) CPU and 16 GB RAM,

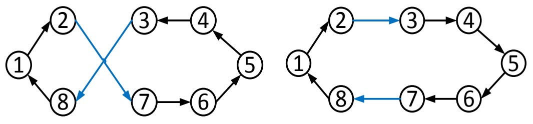


Figure 3. Example of removing an intersection by 2-Opt.

under Microsoft Windows 7 Enterprise as an operating system.

5.1. Structural TSP Instances

To assess the validity of the generated output, we designed and generated synthetic TSP structures with different geometric shapes (circle, square, and triangle). These TSP structures are easier to evaluate than randomized instances. Several instances with different numbers of nodes were generated for each structure, and were input to the HCA algorithm with and without the 2-Opt operation. The percentage difference (*i.e.*, the deviation percentage) between the obtained and the optimal value was calculated as follows:

$$\text{Difference} = \frac{(\text{Obtained Value} - \text{Optimal Value})}{\text{Optimal Value}} \times 100\% \tag{23}$$

In the circular structure, the circle circumference was divided into various numbers of nodes. Note that the number of nodes influences the inter-nodal distance, with fewer nodes increasing the distance between nodes. The node number was varied as 25, 50, 75, 100, 125, and 150. By dividing the circumference of the circle into a specific number of nodes, the first and last nodes will have the same coordinate. The shortest path length was calculated by the circle circumference formula ($2 \times \pi \times r$). The circle was centered at (1, 1) and its diameter was set to 2 (*i.e.*, $r = 1$). Consequently, its circumference was 6.28. The obtained results are reported in Table 2.

As shown in Table 2, the HCA found the shortest path in each instance of this structure, both with and without the 2-Opt operation. The circle instances are relatively easy to solve because the distance decreases with increasing number of nodes. Thus, the soil amount will be reduced more quickly on shorter edges than on longer edges, steering the algorithm towards the shorter edges. Figure 4 shows the output of the HCA on circular TSPs with different numbers of nodes.

Next, the TSP was solved on a square structure. Here, the nodes were evenly spaced in an $N \times N$ grid. The shortest tour distance was the product of the number of nodes and the distance between the nodes (assumed as one unit). For example, in the 16-point (8 × 8) grid, the shortest path was (1 × 16 = 16). For an odd number of nodes, the cost of traveling to the last node was based on the length of the hypotenuse (1.41 in the present examples). Ten instances with different numbers of nodes were generated, and solved by the HCA with and without the 2-Opt operation. The results are listed in Table 3.

As shown in Table 3, the HCA obtained the optimal results (the shortest path) both with and without the 2-Opt operation. The exception was "Square_144", whose solution deviated very slightly from the optimal. The

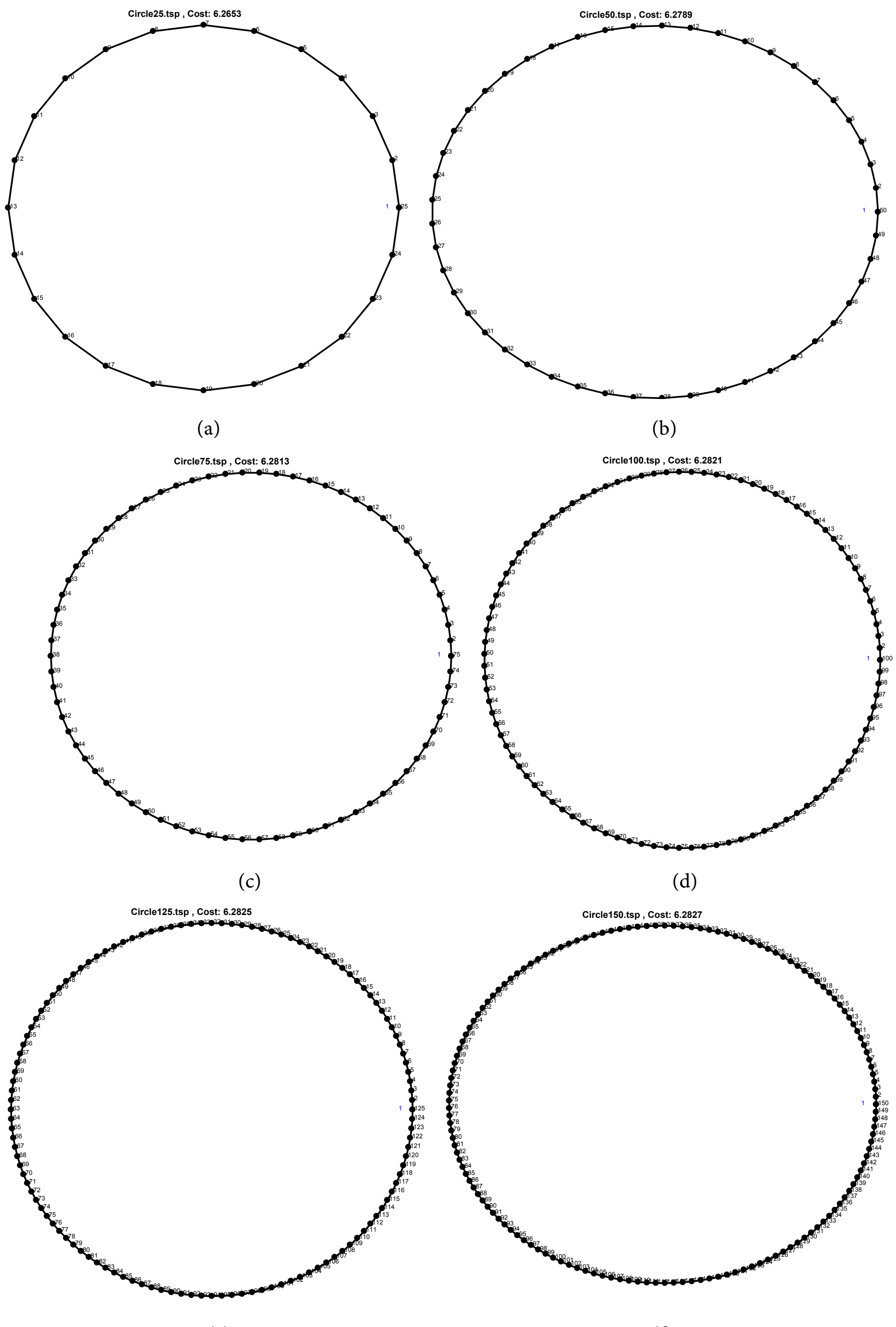


Figure 4. TSP solutions on circular grids. (a) Circle_25, Cost = 6.28; (b) Circle_50, Cost = 6.28; (c) Circle_75, Cost = 6.28; (d) Circle_100, Cost = 6.28; (e) Circle_125, Cost = 6.28; (f) Circle_150, Cost = 6.28.

outputs of HCA with 2-Opt on square grids of different sizes are shown in **Figure 5**.

Finally, the TSP was solved on an equilateral triangular grid. The number of nodes was varied as 9, 25, 49, 81, 121, and 169. **Table 4** lists the obtained results with and without the 2-Opt operation.

Table 2. TSP results on a circular structure.

Instance Name	Optimal Solution	With 2-Opt			Without 2-Opt		
		Result	Avg.	Difference	Result	Avg.	Difference
Circle_25	6.28	6.28	0.72	0%	6.28	0.65	0%
Circle_50	6.28	6.28	4.48	0%	6.28	4.47	0%
Circle_75	6.28	6.28	15.43	0%	6.28	15.13	0%
Circle_100	6.28	6.28	38.60	0%	6.28	37.23	0%
Circle_125	6.28	6.28	80.49	0%	6.28	74.50	0%
Circle_150	6.28	6.28	146.68	0%	6.28	136.99	0%

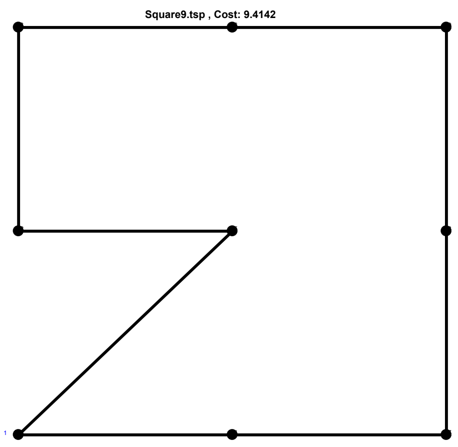
Table 3. TSP results on a square structure.

Instance Name	Optimal Solution	With 2-Opt			Without 2-Opt		
		Result	Avg.	Difference	Result	Avg.	Difference
Square_9	9.41	9.41	0.10	0%	9.41	0.01	0%
Square_16	16	16	0.22	0%	16	0.22	0%
Square_25	25.41	25.41	0.65	0%	25.41	0.64	0%
Square_36	36	36	1.74	0%	36	1.72	0%
Square_49	49.41	49.41	4.35	0%	49.41	4.33	0%
Square_64	64	64	9.90	0%	64	9.55	0%
Square_81	81.41	81.41	20.53	0%	81.41	19.58	0%
Square_100	100	100	39.55	0%	100	39.44	0%
Square_121	121.41	121.41	74.054	0%	121.41	72.38	0%
Square_144	144	144	130.93	0%	146.89	125.18	0.02%

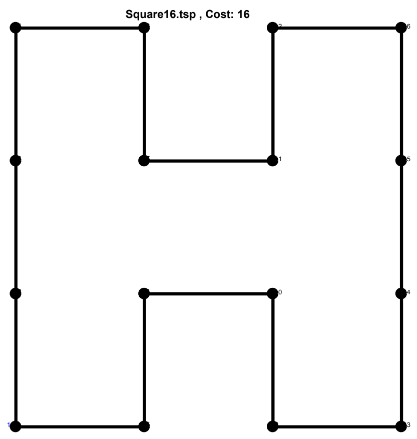
Table 4. TSP results on a triangular structure.

Instance Name	Optimal Solution	With 2-Opt			Without 2-Opt		
		Result	Avg.	Difference	Result	Avg.	Difference
Square_9	9.41	9.41	0.10	0%	9.41	0.01	0%
Square_16	16	16	0.22	0%	16	0.22	0%
Square_25	25.41	25.41	0.65	0%	25.41	0.64	0%
Square_36	36	36	1.74	0%	36	1.72	0%
Square_49	49.41	49.41	4.35	0%	49.41	4.33	0%
Square_64	64	64	9.90	0%	64	9.55	0%
Square_81	81.41	81.41	20.53	0%	81.41	19.58	0%
Square_100	100	100	39.55	0%	100	39.44	0%
Square_121	121.41	121.41	74.054	0%	121.41	72.38	0%
Square_144	144	144	130.93	0%	146.89	125.18	0.02%

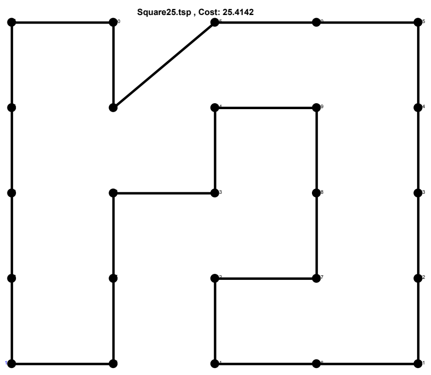
The HCA with and without 2-Opt operation produced almost similar results, except for the triangles with 121 and 169 nodes where using 2-Opt gave better


Square9.tsp , Cost: 9.4142

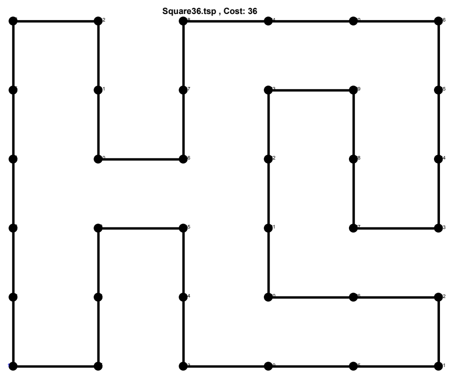
(a)


Square16.tsp , Cost: 16

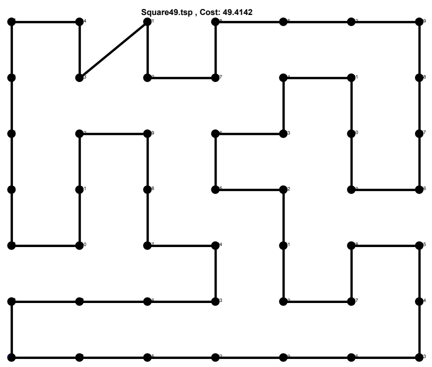
(b)


Square25.tsp , Cost: 25.4142

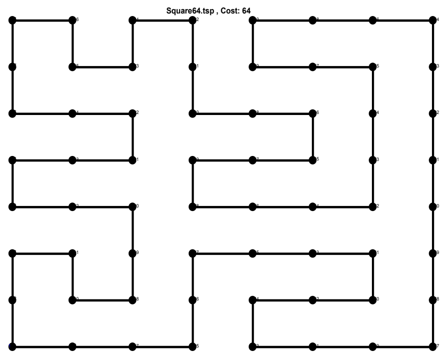
(c)


Square36.tsp , Cost: 36

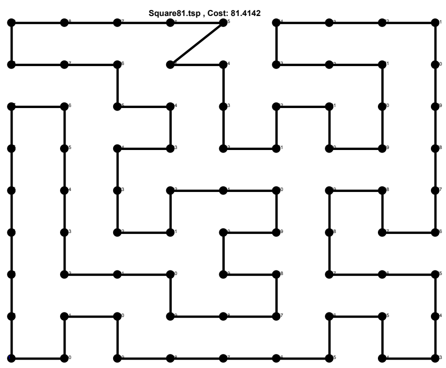
(d)


Square49.tsp , Cost: 49.4142

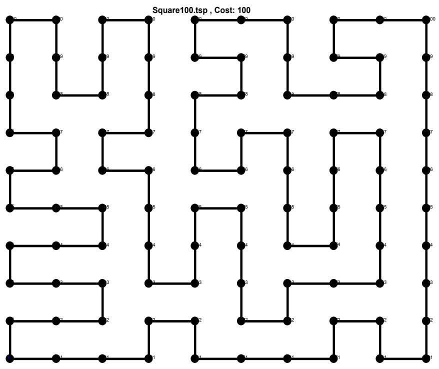
(e)


Square64.tsp , Cost: 64

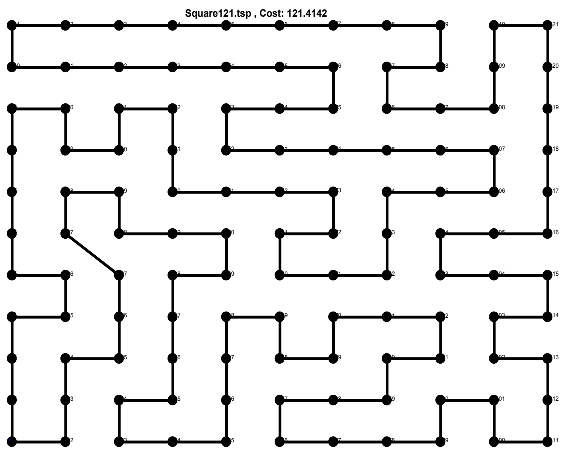
(f)


Square81.tsp , Cost: 81.4142

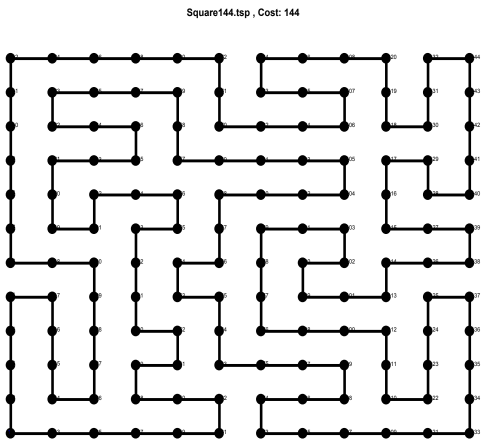
(g)


Square100.tsp , Cost: 100

(h)



(i)



(j)

Figure 5. TSP solutions on square grids. (a) Cost = 9.41; (b) Cost = 16; (c) Cost = 25.41; (d) Cost = 36; (e) Cost = 49.41; (f) Cost = 64; (g) Cost = 81.41; (h) Cost = 100; (i) Cost = 121.41; (j) Cost = 144.

results. The TSP is more difficult on the triangular structure than on the other structures, because many hypotenuses connect the nodes to different layers. The outputs of the HCA using 2-Opt on triangular grids with different node numbers are reported in Figure 6.

The average execution times for solving all the TSP structural instances by HCA are presented by Figure 7. The execution time of the HCA increases with increasing number of nodes because of the information sharing process. With increasing number of nodes the solution space increases exponentially, a defining characteristic of NP-hard problems which also affects execution time. The execution time also largely depends on the implementation of the algorithm, and on the compilers, machines specifications, and operating systems used.

Figure 7 shows that the 2-Opt operation has little effect on the execution time in small instances (problems with a low node count), but noticeably increases the execution time in larger problems. However, 2-Opt was found to improve the quality of the solution for structures with a high number of nodes.

5.2. Benchmark TSP Instances

Next, the HCA was applied to a number of standard benchmark instances from the TSPLIB library [36]. The selected instances have different structures with different numbers of cities. Some of these instances are geographical and based on real city maps; others are based on VLSI applications, drilling, and printed circuit boards. The edge-weights (distances) between the nodes were calculated by the Euclidean distance (Equation (1)), and rounded to integers. The TSP file format is detailed in Reinelt [37]. On the benchmark problems, the HCA was combined with the 2-Opt operation, which was found to improve the solution quality in structural instances with large numbers of nodes. The results are presented in Table 5. In this table, the number in each instance name denotes the number of cities, and the difference column denotes the percentage difference

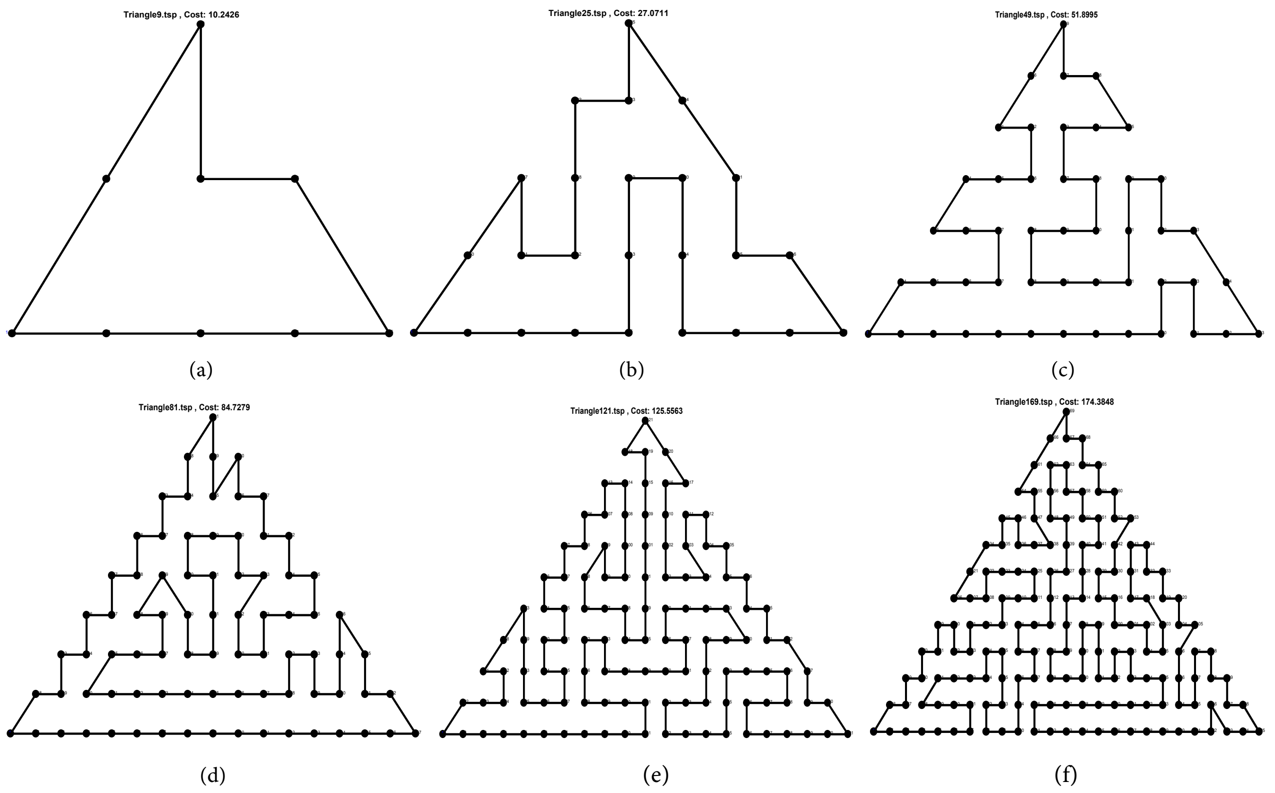

Triangle9.tsp , Cost: 10.2426
Triangle25.tsp , Cost: 27.0711
Triangle49.tsp , Cost: 51.8995
Triangle81.tsp , Cost: 84.7279
Triangle121.tsp , Cost: 125.5563
Triangle169.tsp , Cost: 174.3848
(a)
(b)
(c)
(d)
(e)
(f)

Figure 6. TSP solutions on equilateral triangular grids. (a) Cost = 10.24; (b) Cost = 27.07; (c) Cost = 51.899; (d) Cost = 84.727; (e) Cost = 125.556; (f) Cost = 174.38.

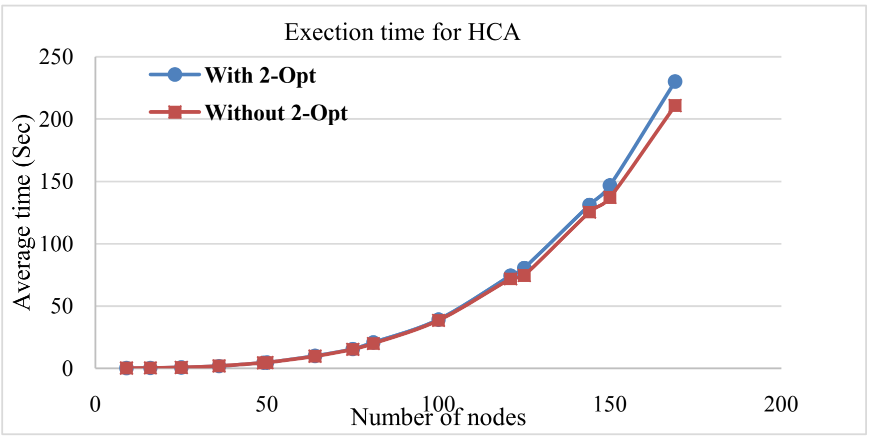


Figure 7. Relationship between HCA execution time and instance size of TSP on circular, square and triangular grids.

from the optimal solution using Equation (7).

Table 5 shows that the HCA achieved a high performance when solving TSP. The HCA found the optimal solution in 20 out of 24 instances, and the differences in the other instances were minor. According to the *P*-value, there is no significant difference between the results. Table 6 reports the minimum, average, and maximum values of the cost, time and iteration number among 10 HCA executions for each instance.

Table 5. HCA results on benchmark TSP instances.

No.	Instance name	Node number	Optimal result	HCA	Difference %
1	berlin52	52	7542	7542	0
2	ch130	130	6110	6110	0
3	ch150	150	6528	6528	0
7	d198	198	15,780	15,780	0
4	eil51	51	426	426	0
5	eil76	76	538	538	0
6	eil101	101	629	629	0
8	kroA100	100	21,282	21,282	0
9	kroA150	150	26,524	26,614	0.00339
10	kroA200	200	29,368	29,368	0
11	kroB100	100	22,141	22,141	0
12	kroB150	150	26,130	26,132	0.00008
13	kroB200	200	29,437	29,455	0.00061
14	kroC100	100	20,749	20,749	0
15	kroD100	100	21,294	21,294	0
16	kroE100	100	22,068	22,068	0
17	lin105	105	14,379	14,379	0
18	pr76	76	108,159	108,159	0
19	pr107	107	44,303	44,303	0
20	pr124	124	59,030	59,030	0
21	pr136	136	96,772	96,861	0.00092
22	rat195	195	2323	2323	0
23	st70	70	675	675	0
24	ts225	225	126,643	126,643	0
	Average		29534.6	29542.9	
	T-test (*P*-value)			0.12174	

The results in Table 6 demonstrate the efficiency and effectiveness of the HCA algorithm. In particular, the average result and optimal solution are very close in all instances. The maximum difference was 0.00823% on the kroA150 benchmark, and zero on the pr124 benchmark. Moreover, the HCA optimized the solution on most benchmarks within a few iterations. This early convergence is attributed to information sharing among the water drops, and the use of the 2-Opt operation in the condensation stage. The solutions to the benchmark instances are displayed in the Figure S1 (**Appendix**).

The minimal cost in HCA was compared with the reported results of other water-based algorithms, namely, the intelligent water drops (IWD) algorithm and its modifications, water wave optimization (WWO), the water flow-like

Table 6. Minimum, average, and maximum HCA results on benchmark TSP instances.

Instance name		Cost	Time(s)	#Iteration	Instance name		Cost	Time(s)	Iteration
	Min	7542	5.15	5		Min	29,455	455.53	35
berlin52	Avg.	7565.3	5.36	37.9	kroB200	Avg.	29519.9	464.62	200.6
	Max	7758	5.78	55		Max	29,612	474.95	305
	Min	6110	93.08	53		Min	20,749	39.54	11
ch130	Avg.	6128.9	95.79	168.2	kroC100	Avg.	20,751	39.74	71
	Max	6177	101.61	359		Max	20,769	39.96	303
	Min	6528	149.98	17		Min	21,294	40.10	5
ch150	Avg.	6550.8	157.55	154.4	kroD100	Avg.	21416.4	40.43	151.4
	Max	6570	162.05	347		Max	21,772	40.79	299
	Min	15780	415.00	47		Min	22,068	40.27	23
d198	Avg.	15785.3	422.04	209.6	kroE100	Avg.	22152.9	40.63	107
	Max	15,794	432.66	593		Max	22,389	41.07	203
	Min	426	4.70	11		Min	14,379	46.59	11
eil51	Avg.	426.85	4.73	47.2	lin105	Avg.	14385.6	47.25	111.8
	Max	430	4.79	86		Max	14,412	47.75	263
	Min	538	16.19	11		Min	108,159	16.47	5
eil76	Avg.	538.5	16.34	47.8	pr76	Avg.	108163.3	16.58	32
	Max	539	16.45	137		Max	108,202	16.84	215
	Min	629	41.37	34		Min	44,303	48.25	5
eil101	Avg.	632	41.60	99.9	pr107	Avg.	44367.5	48.84	80
	Max	638	41.85	274		Max	44,438	49.35	293
	Min	21,282	40.39	23		Min	59,030	78.60	5
kroA100	Avg.	21308.1	40.70	112.4	pr124	Avg.	59,030	79.19	47
	Max	21,369	41.03	275		Max	59,030	79.96	101
	Min	26,614	161.42	11		Min	96,861	109.96	41
kroA150	Avg.	26742.2	162.67	204.2	pr136	Avg.	96985.1	110.93	204.2
	Max	26,917	163.69	371		Max	97,235	113.14	371
	Min	29,368	461.13	29		Min	2323	385.41	29
kroA200	Avg.	29396.3	469.99	150.2	rat195	Avg.	2334.6	390.44	314.6
	Max	29,518	479.46	299		Max	2343	396.57	557
	Min	22,141	39.72	5		Min	675	12.42	11
kroB100	Avg.	22,222	40.00	19.4	st70	Avg.	676.5	12.59	77.2
	Max	22,258	40.38	101		Max	681	12.71	182
	Min	26,132	157.96	83		Min	126,643	626.73	125
kroB150	Avg.	26216.2	161.33	217.4	ts225	Avg.	126788.1	636.24	336.2
	Max	26,329	165.14	419		Max	126,962	643.72	647

algorithm (WFA), and river formation dynamics (RFD). The comparisons are summarized in **Table 7**. The results of the original and a modified IWD (columns 4 and 5, respectively) were taken from [1] and from [38], respectively. The results of another modified IWD, called the exponential ranking selection IWD (ERS-IWD; column 6), were extracted from [3]. The results of columns 7 and 8 were taken from [5], who implemented the IWD and their proposed adaptive IWD on TSP instances. The WWO results (column 9) were taken from [7]. The WFA and RFD results (columns 10 and 11) were borrowed from [8] and from [9], respectively. The best results are marked in bold font.

The numbers of instances solved by these algorithms are insufficient for calculating an accurate *P*-value statistic. Moreover, some of these algorithms perform as well as HCA in certain instances. However, as confirmed in **Table 7**, HCA outperforms the original IWD algorithm and its various modifications. One plausible reason for the poor performance of the IWD algorithm is the premature convergence and stagnation in local optimal solutions. In contrast, HCA can escape from local optima by exploiting the depths of the paths along with the soil amount. These actions diversify the solutions. The most competitive opponent to HCA was WFA, which also optimized the solutions in the tested instances. In contrast, the WWO performed poorly because this algorithm

Table 7. Best results of HCA, the original IWD, modified IWDs, WWO, WFA, and RFD.

Instance name	Optimal result	HCA	Original IWD (4)	IWD (5)	ERS-IWD (6)	Adaptive IWD		WWO (9)	WFA (10)	RFD (11)
						IWD (7)	AIWD (8)			
berlin52	7542	7542	-	7542	-	-	-	-	-	-
ch130	6110	6110	-	-	6316	-	-	6338	6110	-
ch150	6528	6528	-	-	-	-	-	7014	6528	-
eil51	426	426	471	426	429	434	426	427	426	441.9
eil76	538	538	559	540	545	552	538	557	538	-
eil101	629	629	-	639	654	-	-	-	629	-
kroA100	21,282	21,282	23,156	21,429	21,959	23,183	21,304	21,668	21,282	-
kroA150	26,524	26,614	-	-	-	-	-	-	26,524	-
kroA200	29,368	29,368	-	-	31,680	-	-	31,064	29,368	-
kroC100	20,749	20,749	-	20,816	-	-	-	-	-	-
lin105	14,379	14,379	-	14,393	14,696	-	-	-	-	-
pr76	108,159	108,159	-	109,608	-	-	-	-	-	-
rat195	2323	2323	-	-	-	2461	2338			-
st70	675	675	-	676	-	710	675	-	-	-
ts225	126,643	126,643	-	-	-	275791	127325	-	-	-
Average	24791.7	24797.7	8062	19563.2	10,897	50521.8	25434.3	11178	11426	441.9
P-values vs HCA			0.4025	0.2701	0.1598	0.3219	0.1878	0.1302	0.2813	-

was originally designed for continuous-domain problems, and its operations need adjustment for combinatorial problems. Moreover, the WWO adopts a reducing population-size strategy, which degrades its performance in some problems. Finally, the WWO suffers from slow convergence because it depends only on the altitude of the nodes.

The performances of HCA, IWD, adaptive IWD (AIWD) and modified IWD (MIWD) are further compared in **Table 8**. The best and average results of IWD and AIWD were taken from [5], while those of MIWD were taken from [6].

This comparison aims to compare the robustness of HCA and other algorithms. Despite there being no significant differences between the results (best, average), the average results are closer to the optimal in HCA than in the other algorithms, suggesting the superior robustness of HCA. **Table 9** compares the runtimes of the HCA, IWD and AIWD. The best and average execution times and iteration numbers of the IWD algorithms were taken from [5].

According to **Table 9**, HCA reaches the best solution after fewer iterations than IWD and AIWD. This result confirms the superior efficiency of HCA. Moreover, adding the other stages of the water cycle did not affect the average execution time of HCA. **Figure 8** plots the average execution times of the three algorithms implemented on five benchmark problems.

Optimal-solution searching by HCA was compared with those of other well-known algorithms, namely, an ACO algorithm combined with fast opposite gradient search (FOGS-ACO) [39], a genetic simulated annealing ant colony system with PSO (GSAACS-PSO) [40], an improved discrete bat algorithm (IBA) [27], set-based PSO (S-CLPSO) [41], a modified discrete PSO with a newly introduced mutation factor C3 (C3D-PSO); results taken from [23], an adaptive simulated annealing algorithm with greedy search (ASA-GS) [11], the firefly algorithm (FA) [42], a hybrid ACO enhanced with dual NN (ACOMAC-DNN) [43], a discrete PSO (DPSO) [26], a self-organizing neural network using the immune system (ABNET-TSP) [44], and an improved discrete cuckoo search algorithm (IDCS) [45]. **Table 10** summarizes the comparison results.

Table 8. Best and average results of HCA, IWD, AIWD, and MIWD.

Instance Name	HCA		IWD		AIWD		MIWD	
	Best	Avg.	Best	Avg.	Best	Avg.	Best	Avg.
Eil51	426	426.85	434	443.2	426	428.4	428.98	432.62
St70	675	676.5	710	724.93	675	682.5	677.1	684.08
Eil76	538	538.5	552	564.43	538	542.86	549.96	558.23
KroA100	21282	21308.1	23183	23548.37	21,304	21586.73	21407.57	21904.03
rat195	2323	2334.6	2461	2480.6	2338	2347.8	-	-
ts225	126643	126788.1	755791	276140.75	127,325	128323.5	-	-
Average	29912.8	29947.6	130521.8	50650.4	25434.3	25652.0	5765.9	5894.7
T-test (*P*-values) vs HCA			0.3722	0.3656	0.3570	0.2867	0.3209	0.3611

Table 9. Average execution times and best and average iteration numbers in HCA, IWD, and Adaptive IWD.

Instance name	HCA		IWD		Adaptive IWD	
	Avg. Time (s)	Iteration [Best, Avg.]	Avg. Time (s)	Iteration [Best, Avg.]	Avg. Time (s)	Iteration [Best, Avg.]
eil51	4.73	[57, 47.2]	154.537	[1509, 3000]	180.648	[190, 3000]
st70	12.59	[83, 77.2]	434.193	[960, 3,500]	453.631	[1769, 3500]
eil76	16.34	[46, 47.8]	567.208	[2147, 3,500]	571.251	[752, 3500]
kroA100	40.7	[89, 112.4]	1364.979	[3698, 3750]	1365.752	[2397, 3750]
rat195	390.44	[401, 314.6]	2023.162	[604, 5000]	2335.9392	[4995, -]
ts225	636.24	[365, 336.2]	3969.892	[1, 5000]	4162.92	[3850, 5000]
Average	142.1		1419.0		1511.7	
T-test (*P*-value) for Avg. Time			0.1162		0.1200	

Table 10. Best results obtained by HCA and other optimization algorithms.

Instance name	Optimal result	HCA	FOGS-ACO	GSAACS-PSO	IBA	S-CLPSO	C3D-PSO	ASA-GS	FA	ACOMAC-DNN	DPSO	ABNET-TSP	IDCS
berlin52	7542	7542	7546.6	7542	7542	7542		7544.7	7544.36	-	7542	7542	7542
ch130	6110	6110	-	6141	-	-		6110.7	-	-	-	6145	6110
ch150	6528	6528	-	6528	-	-		6530.9	-	-	-	6602	6528
eil51	426	426	426	427	426	426	426	428.87	428.87	430.01	427	427	426
eil76	538	538	546.83	538	539	538	538	544.37		552.61	546	541	538
eil101	629	629	633.40	630	634	629		640.21			-	638	629
d198	15,780	15,780		-	-	15,809		15830.6		15,955.6	-		15,781
kroA100	21,282	21,282	22,414	21,282	21,282	21,282	21,282	21285.4	21285.4	21,408.2	-	21,333	21,282
kroA150	26,524	26,614	-	26,524	-	26,537		26524.9		-	-	26,678	26,524
kroA200	29,368	29,368	29,717	29383	-	29,399		29411.5		-	-	29,600	29,382
kroB100	22,141	22,141	-	22141	22,140*	-		22139.1	22139.1	-	-	22,343	22,141
kroB150	26,130	26,132	-	26130	-	-		26140.7	-	-	-	26,264	26,130
kroB200	29,437	29,455	-	29541	-	-		29504.2	-	-	-	29,637	29,448
kroC100	20,749	20,749	-	20,749	20,749	20824.6		20750.8	-	-	-	20,915	20,749
kroD100	21,294	21,294	-	21,309	21,294	21405.6		21294.3	-	-	-	21,374	21,294
kroE100	22,068	22,068	-	22,068	22,068	-		22106.3	-	-	-	22,395	22,068
lin105	14,379	14,379	-	14,379	-	14379		14383	14383	-	-	14,379	14,379
pr76	108,159	108,159	108,864	-	-	108159		108159		-	108280	-	108,159
pr107	44,303	44,303	-	-	44,303			44301.7*	44346	-	-	-	44,303
pr124	59,030	59,030	-	-	59,030			59030.7	59030	-	-	-	59,030
pr136	96,772	96,861	-	-	97,547			96966.3	97182.7	-	-	-	96,790
rat195	2323	2323	-	-	-			2345.2		-	-	-	2324
st70	675	675	678.93	-	675	675	675	677.11	677.11	-	675	-	675
ts225	126,643	126,643	-	-	-	-		126646	-	-	-	-	126,643
Average	29534.6	29542.9	21353.3	16062.2	24479.2	20585.0	5730	29,554	29,669	9587	23,494	16,051	29536.5
P-values versus HCA			0.1120	0.6755	0.3327	0.3088	-	0.1129	0.2684	0.1536	0.3360	0.0014	0.1890

* Incorrect.

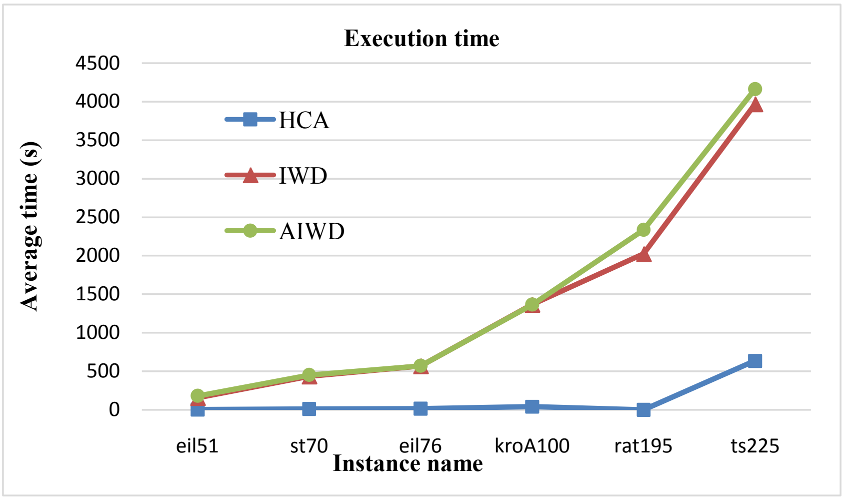


Figure 8. Average execution times of HCA, IWD, and Adaptive IWD.

Although the complexity of the TSP increases with increasing number of cities, the HCA outperformed the other algorithms in most instances. The *P*-values indicate there are no significant differences between HCA and other algorithms, except between HCA and ABNET-TSP, where the HCA was better. The HCA competed with other algorithms such as the IDCS algorithm; indeed, the results of HCA and IDCS were not noticeably different even for large problems. The high performance of HCA was again attributed to the effective design of the HCA and that included an information sharing process among the water drops. This process helps the HCA exploit the promising solutions and increases the speed of algorithm convergence. The additional stages of the HCA assist with exploring different solutions (enhancing the search capability), and prevent trapping in local optima.

5.3. HCA Convergence Evaluation

This section analyses the performance of the HCA and its convergence rate. As previously stated, the maximum iteration number was set to three times the number of nodes in the instance. Figure 9 shows the convergence of the algorithm on the berlin52 instance. The cost along the Y-axis denotes the total route length.

According to Figure 9, the solution was optimized after 65 iterations. The berlin52 benchmark is relatively easy to solve because the node distribution reduces the possibility of falling into local optima. The local and global solutions are the best solution at the end of each iteration and the best solution among all iterations, respectively. Note that the algorithm converges towards the optimal solution. In addition, the HCA generated different solutions in every iteration and the search process was prevented from stagnating by the depth factor and the information sharing among the water drops. The depth factor increases the chance of selecting previously unexplored or little-used paths. Figure 10 shows the convergence of the algorithm on the eil51 instance. The solution was

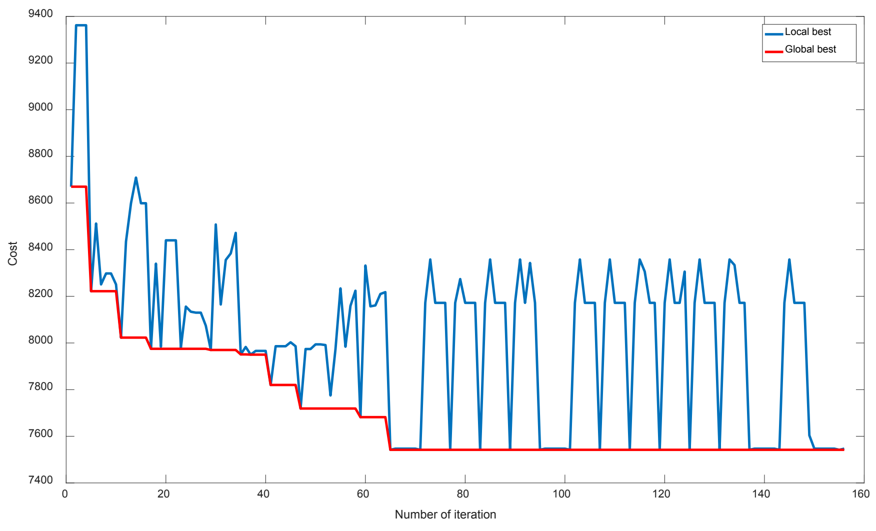


Figure 9. Local (blue) and global (red) best solutions on berlin52.

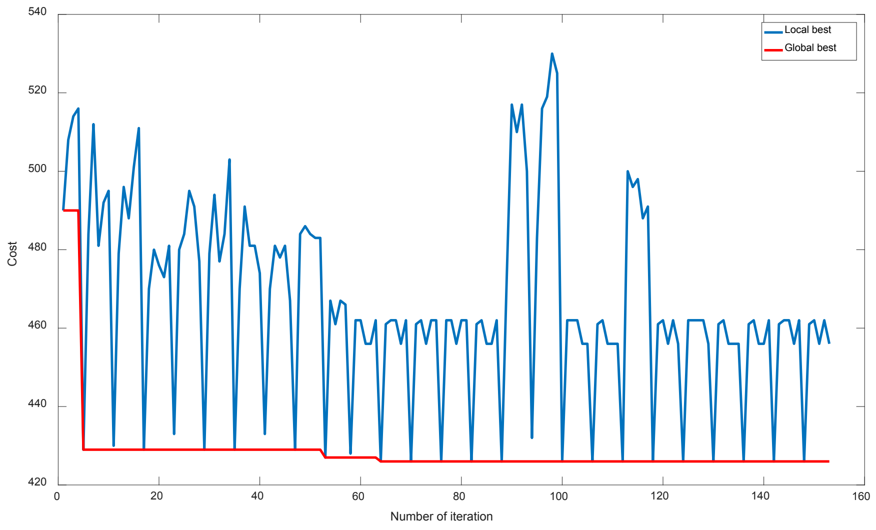


Figure 10. Local (blue) and global (red) best solutions on eil51.

optimized at the 64th iteration.

Figure 11 illustrates the convergence behavior of the HCA on the eil67 instance. Here, the solution was optimized at iteration 171.

Figure 12 illustrates the convergence behavior of the HCA on the eil101 instance. The optimal solution was found at iteration 99. Moreover, the smooth convergence rate confirms the good balance between the exploration and exploitation processes.

Figure 13 shows the convergence of the global best solution on the st70 instance. The solution was optimized at iteration 125.

In summary, the convergence rate of the HCA proves the effectiveness of the algorithm design. Furthermore, the algorithm searches the optimal solution until the final iterations, without stagnation in local optima. It also converges rapidly on easy instances.

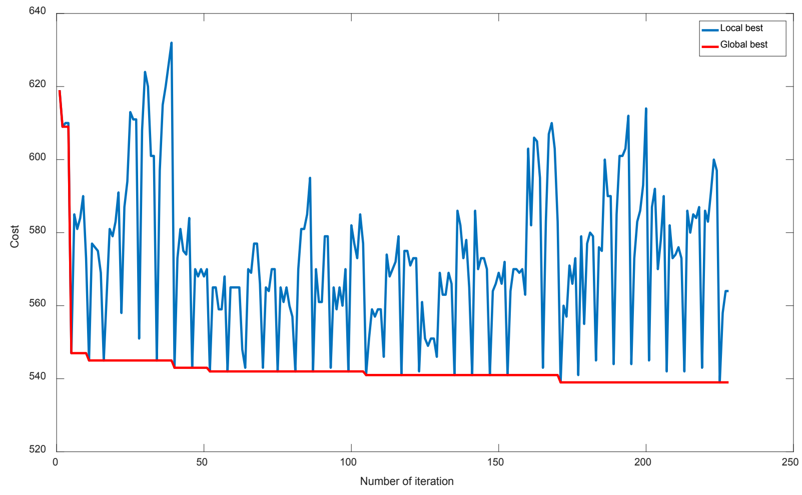


Figure 11. Local (blue) and global (red) best solutions on eil76.

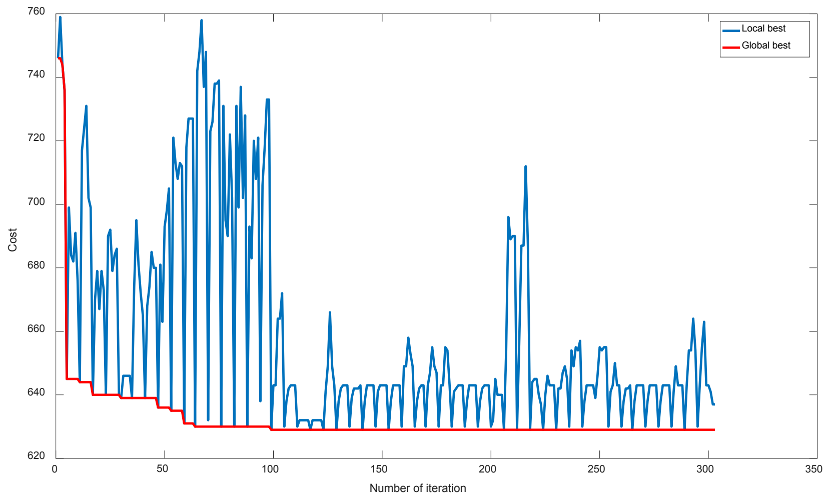


Figure 12. A graph for local vs. global solution on eil101.

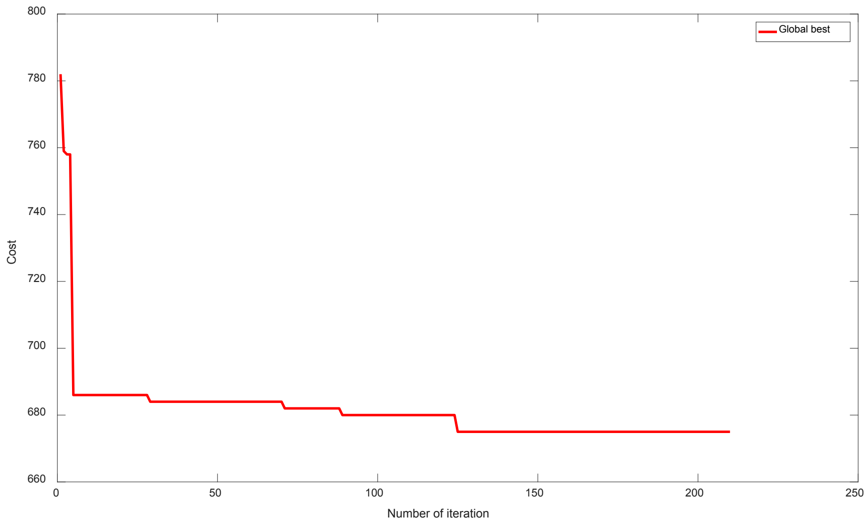


Figure 13. Global best solution on st70.

6. Conclusions

In this paper, HCA was applied on an archetypal NP-hard problem (the TSP). Initially, the performance of the algorithm was tested on simple geometric structures which are easy to design and understand. Parameter tuning was also performed on these structures. The obtained results indicate the flexibility and capability of the algorithm in solving such problems. Moreover, the algorithm provided different same-cost solutions to the same problem. This validates the effective design of the exploration and exploitation processes of the algorithm. The geometric TSP instances are useful for evaluating other new algorithms due to their simple design, and different shapes can be designed by the same principle.

Next, the algorithm was tested on various standard benchmarks taken from the literature. The algorithm provided high-quality solutions and outperformed other metaheuristic algorithms in seeking the minimum path. Also, the HCA found the optimal solution within a few iterations. The HCA showed its ability to escape from local optima and find the global solution. The strong optimization capability of the HCA is conferred by the efficient design of the exploration and exploitation processes. Moreover, by utilizing both direct and indirect communication to share information among the water drops, the algorithm steers towards better solutions within a small number of iterations and helps to diversify the search space. Significance figures show that, at the very least, HCA is no worse than other algorithms. The added advantage of HCA is that all stages of the hydrological water cycle are included, leading to an overall conceptual framework under which other water-based algorithms can be placed. In addition, the inclusion of all stages allows both direct and indirect communication to take place among particles, leading to enhanced swarm intelligence.

In summary, the HCA demonstrated strong performance in structural and benchmark TSP instances. It obtained the optimal solution in most instances, confirming the effectiveness of the algorithm framework. Therefore, the HCA structure is a feasible approach for solving TSPs. The HCA tends to fully explore the graph, providing diverse solutions at fast convergence speeds. Also, as confirmed by the convergence behavior of the algorithm, the HCA successfully avoids potential stagnation in local optima.

The HCA performance could additionally be investigated on asymmetric TSP instances. Although the HCA optimizes the TSP solution within a reasonable timeframe, further enhancements would reduce its execution time on large instances. Furthermore, the HCA can be used for solving other NP-hard optimization problems.

References

[1] Shah-Hosseini, H. (2007) Problem Solving by Intelligent Water Drops. 2007 *IEEE Congress on Evolutionary Computation*, **1**, 3226-3231. https://doi.org/10.1109/CEC.2007.4424885

[2] Eskandar, H., Sadollah, A., Bahreininejad, A. and Hamdi, M. (2012) Water Cycle Algorithm—A Novel Metaheuristic Optimization Method for Solving Constrained Engineering Optimization Problems. *Computers & Structures*, **110-111**, 151-166. https://doi.org/10.1016/j.compstruc.2012.07.010

[3] Alijla, B.O., Wong, L.-P., Lim, C.P., Khader, A.T. and Al-Betar, M.A. (2014) A Modified Intelligent Water Drops Algorithm and Its Application to Optimization Problems. *Expert Systems with Applications*, **41**, 6555-6569. https://doi.org/10.1016/j.eswa.2014.05.010

[4] Wedyan, A., Whalley, J. and Narayanan, A. (2017) Hydrological Cycle Algorithm for Continuous Optimization Problems. *Journal of Optimization*, **2017**, Article ID: 3828420. https://doi.org/10.1155/2017/3828420

[5] Msallam, M.M. and Hamdan, M. (2011) Improved Intelligent Water Drops Algorithm Using Adaptive Schema. *International Journal of Bio-Inspired Computing*, **3**, 103. https://doi.org/10.1504/IJBIC.2011.039909

[6] Shah-Hosseini, H. (2009) The Intelligent Water Drops Algorithm: A Nature-Inspired Swarm-Based Optimization Algorithm. *International Journal of Bio-Inspired Computing*, **1**, 71-79. https://doi.org/10.1504/IJBIC.2009.022775

[7] Wu, X.-B., Liao, J. and Wang, Z.-C. (2015) Water Wave Optimization for the Traveling Salesman Problem. 11*th International Conference of Intelligent Computing Theories and Methodologies*, Fuzhou, 20-23 August 2015, 137-146. https://doi.org/10.1007/978-3-319-22180-9_14

[8] Srour, A., Othman, Z.A. and Hamdan, A.R. (2014) A Water Flow-Like Algorithm for the Travelling Salesman Problem. *Advances in Electrical and Computer Engineering*, **2014**, 1-14. https://doi.org/10.1155/2014/436312

[9] Rabanal, P., Rodríguez, I. and Rubio, F. (2009) Applying River Formation Dynamics to Solve NP-Complete Problems. In: Chiong, R., Ed., *Nature-Inspired Algorithms for Optimisation*, Springer, Berlin Heidelberg, 333-368. https://doi.org/10.1007/978-3-642-00267-0_12

[10] Zhan, S., Lin, J., Zhang, Z. and Zhong, Y. (2016) List-Based Simulated Annealing Algorithm for Traveling Salesman Problem. *Computational Intelligence and Neuroscience*, **2016**, 1-12. https://doi.org/10.1155/2016/1712630

[11] Geng, X., Chen, Z., Yang, W., Shi, D. and Zhao, K. (2011) Solving the Traveling Salesman Problem Based on an Adaptive Simulated Annealing Algorithm with Greedy Search. *Applied Soft Computing*, **11**, 3680-3689. https://doi.org/10.1016/j.asoc.2011.01.039

[12] Braun, H. (1991) On Solving Travelling Salesman Problems by Genetic Algorithms. In: *Parallel Problem Solving from Nature*, Springer-Verlag, Berlin/Heidelberg, 129-133. https://doi.org/10.1007/BFb0029743

[13] Grefenstette, J., Gopal, R., Rosmaita, B. and Van Gucht, D. (1985) Genetic Algorithms for the Traveling Salesman Problem. *Proceedings of the First International Conference on Genetic Algorithms*, Pittsburgh, July 1985, 160-168.

[14] Larranaga, P., Kuijpers, C.M.H., Murga, R.H., Inza, I. and Dizdarevic, S. (1999) Genetic Algorithms for the Travelling Salesman Problem: A Review of Representations and Operators. *Artificial Intelligence Review*, **13**, 129-170. https://doi.org/10.1023/A:1006529012972

[15] Moorthy, S.K. (2012) Evolving Optimal Solutions by Nature Inspired Algorithms. PhD Thesis, University of Pune, Pune.

[16] Potvin, J.-Y. (1996) Genetic Algorithms for the Traveling Salesman Problem. *Annals of Operations Research*, **63**, 337-370. https://doi.org/10.1007/BF02125403

[17] Rao, A. and Hedge, S. (2015) Literature Survey on Travelling Salesman Problem Using Genetic Algorithms. *International Journal of Advanced Research in Education Technology*, **2**, 42.

[18] Vaishnav, P., Choudhary, N. and Jain, K. (2017) Traveling Salesman Problem Using Genetic Algorithm: A Survey. *International Journal of Scientific Research in Computer Science*, *Engineering and Information Technology*, **2**, 105-108.

[19] Dorigo, M. and Gambardella, L.M. (1997) Ant Colonies for the Travelling Salesman Problem. *BioSystems*, **43**, 73-81. https://doi.org/10.1016/S0303-2647(97)01708-5

[20] Stutzle, T. and Hoos, H.H. (1997) MAX-MIN Ant System and Local Search for the Traveling Salesman Problem. *Proceedings of* 1997 *IEEE International Conference on Evolutionary Computation*, Indianapolis, 13-16 April 1997, 309-314. https://doi.org/10.1109/ICEC.1997.592327

[21] Gambardella, L.M. and Dorigo, M. (1996) Solving Symmetric and Asymmetric TSPs by Ant Colonies. *Proceedings of IEEE International Conference on Evolutionary Computation*, Nagoya, 20-22 May 1996, 622-627. https://doi.org/10.1109/ICEC.1996.542672

[22] Hlaing, Z.C.S.S. and Khine, M.A. (2011) An Ant Colony Optimization Algorithm for Solving Traveling Salesman Problem. *International Conference on Information Communication and Management*, **16**, 54-59.

[23] Zhong, W., Zhang, J. and Chen, W. (2007) A Novel Discrete Particle Swarm Optimization to Solve Traveling Salesman Problem. 2007 *IEEE Congress on Evolutionary Computation*, Singapore, 25-28 September 2007, 3283-3287. https://doi.org/10.1109/CEC.2007.4424894

[24] Wang, X., Mu, A. and Zhu, S. (2013) ISPO: A New Way to Solve Traveling Salesman Problem. *Intelligent Control and Automation*, **4**, 122-125. https://doi.org/10.4236/ica.2013.42017

[25] Wang, K.-P., Huang, L., Zhou, C.-G. and Pang, W. (2003) Particle swarm optimization for traveling salesman problem. *Proceedings of the* 2003 *International Conference on Machine Learning and Cybernetics*, Xi'an, 2-5 November 2003, 1583-1585. https://doi.org/10.1109/ICMLC.2003.1259748

[26] Shi, X.H., Liang, Y.C., Lee, H.P., Lu, C. and Wang, Q.X. (2007) Particle Swarm Optimization-Based Algorithms for TSP and Generalized TSP. *Information Processing Letters*, **103**, 169-176. https://doi.org/10.1016/j.ipl.2007.03.010

[27] Osaba, E., Yang, X.-S., Diaz, F., López-Garcia, P. and Carballedo, R. (2016) An Improved Discrete Bat Algorithm for Symmetric and Asymmetric Traveling Salesman Problems. *Engineering Applications of Artificial Intelligence*, **48**, 59-71. https://doi.org/10.1016/j.engappai.2015.10.006

[28] Saji, Y., Riffi, M.E. and Ahiod, B. (2014) Discrete Bat-Inspired Algorithm for Travelling Salesman Problem. 2014 *Second World Conference on Complex Systems*, Agadir, 10-12 November 2014, 28-31. https://doi.org/10.1109/ICoCS.2014.7060983

[29] Basu, S. (2012) Tabu Search Implementation on Traveling Salesman Problem and Its Variations: A Literature Survey. *American Journal of Operations Research*, **2**, 163-173. https://doi.org/10.4236/ajor.2012.22019

[30] Held, M., Hoffman, A.J., Johnson, E.L. and Wolfe, P. (1984) Aspects of the Traveling Salesman Problem. *IBM Journal of Research and Development*, **28**, 476-486. https://doi.org/10.1147/rd.284.0476

[31] Hoffman, K.L. and Padberg, M. (2013) Traveling Salesman Problem. In: *Encyclopedia of Operations Research and Management Science*, Kluwer Academic Publishers,

Dordrecht, 849-853. https://doi.org/10.1007/978-1-4419-1153-7_1068

[32] Li, M., Chen, X., Li, X., Ma, B. and Vitanyi, P.M.B. (2004) The Similarity Metric. *IEEE Transactions on Information Theory*, **50**, 3250-3264. https://doi.org/10.1109/TIT.2004.838101

[33] Helsgaun, K. (2000) An Effective Implementation of the Lin-Kernighan Traveling Salesman Heuristic. *European Journal of Operational Research*, **126**, 106-130. https://doi.org/10.1016/S0377-2217(99)00284-2

[34] Helsgaun, K. (2006) An Effective Implementation of *K*-opt Moves for the Lin-Kernighan TSP Heuristic. Roskilde University, Roskilde, 109.

[35] Rocki, K. and Suda, R. (2012) Accelerating 2-opt and 3-opt Local Search Using GPU in the Travelling Salesman Problem. 2012 12*th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, Ottawa, 13-16 May 2012, 705-706. https://doi.org/10.1109/CCGrid.2012.133

[36] Reinelt, G. (1995) TSPLIB. https://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/

[37] Reinelt, G. (1991) TSPLIB—A Traveling Salesman Problem Library. *ORSA Journal on Computing*, **3**, 376-384. https://doi.org/10.1287/ijoc.3.4.376

[38] Gülcü, S.D., Gülcü, Ş., Kahramanli, H., Campus, A.K. and Selçuklu, K. (2013) Solution of Travelling Salesman Problem Using Intelligent Water Drops Algorithm. *Proceedings of the* 2*nd International Conference on Information Technology and Computer Networks*, Montreal, 27-28August 2016.

[39] Saenphon, T., Phimoltares, S. and Lursinsap, C. (2014) Combining New Fast Opposite Gradient Search with Ant Colony Optimization for Solving Travelling Salesman Problem. *Engineering Applications of Artificial Intelligence*, **35**, 324-334. https://doi.org/10.1016/j.engappai.2014.06.026

[40] Chen, S.-M. and Chien, C.-Y. (2011) Solving the Traveling Salesman Problem Based on the Genetic Simulated Annealing Ant Colony System with Particle Swarm Optimization Techniques. *Expert Systems with Applications*, **38**, 14439-14450. https://doi.org/10.1016/j.eswa.2011.04.163

[41] Chen, W.-N., Zhang, J., Chung, H.S.-H., Zhong, W.-L., Wu, W.-G. and Shi, Y. (2010) A Novel Set-Based Particle Swarm Optimization Method for Discrete Optimization Problems. *IEEE Transactions on Evolutionary Computation*, **14**, 278-300. https://doi.org/10.1109/TEVC.2009.2030331

[42] Wang, M., Fu, Q., Tong, N., Li, M. and Zhao, Y. (2016) An Improved Firefly Algorithm for Traveling Salesman Problems. *Proceedings of the* 2015 4*th National Conference on Electrical, Electronics and Computer Engineering*, Xi'an, 12-13 December 2015.

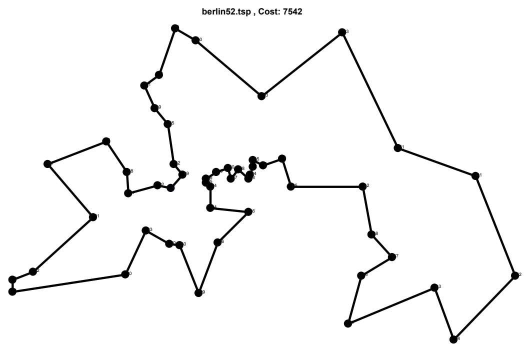
[43] Tsai, C.-F., Tsai, C.-W. and Tseng, C.-C. (2004) A New Hybrid Heuristic Approach for Solving Large Traveling Salesman Problem. *Information Sciences*, **166**, 67-81. https://doi.org/10.1016/j.ins.2003.11.008

[44] Masutti, T.A.S. and de Castro, L.N. (2009) A Self-Organizing Neural Network Using Ideas from the Immune System to Solve the Traveling Salesman Problem. *Information Sciences*, **179**, 1454-1468. https://doi.org/10.1016/j.ins.2008.12.016

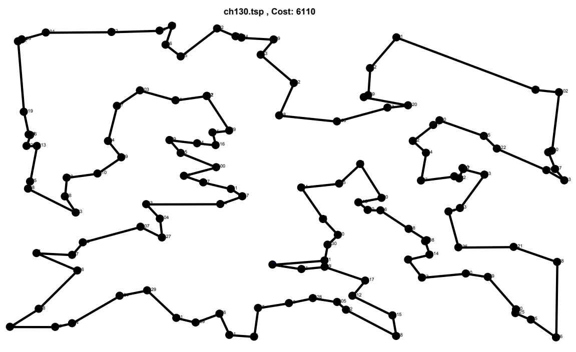
[45] Ouaarab, A., Ahiod, B. and Yang, X.-S. (2014) Discrete Cuckoo Search Algorithm for the Travelling Salesman Problem. *Neural Computing and Applications*, **24**, 1659-1669. https://doi.org/10.1007/s00521-013-1402-2

Appendix

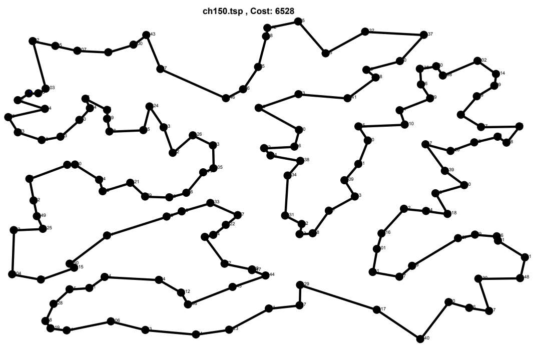
This appendix provides the outputs of HCA when applied on the benchmark instances.



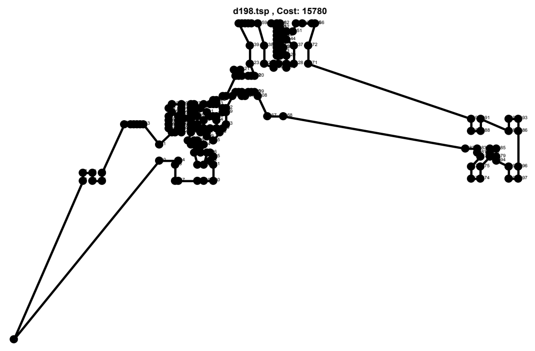
(a)



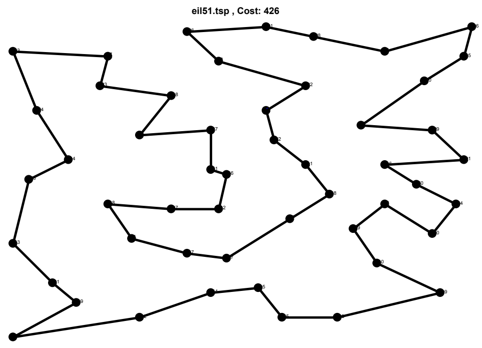
(b)



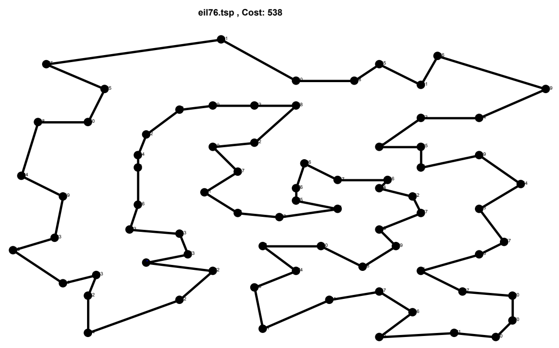
(c)



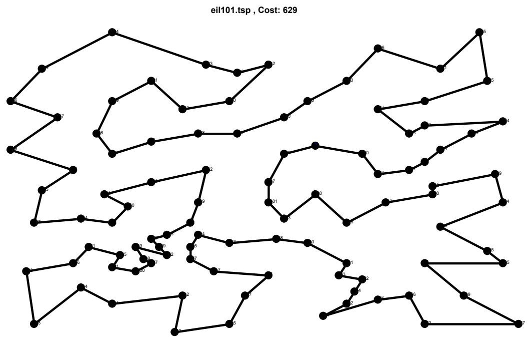
(d)



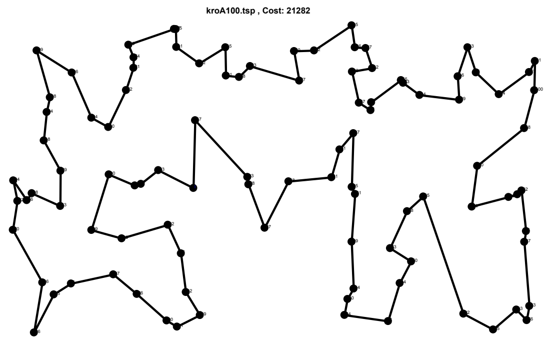
(e)



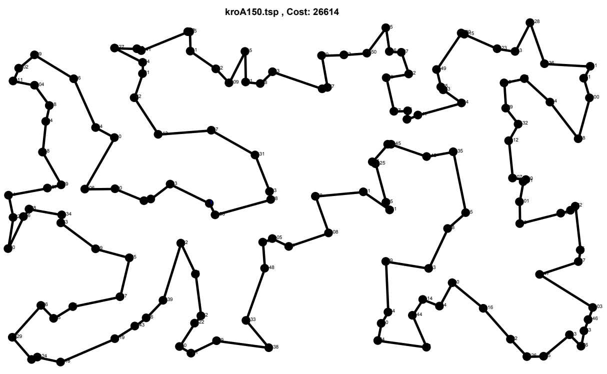
(f)



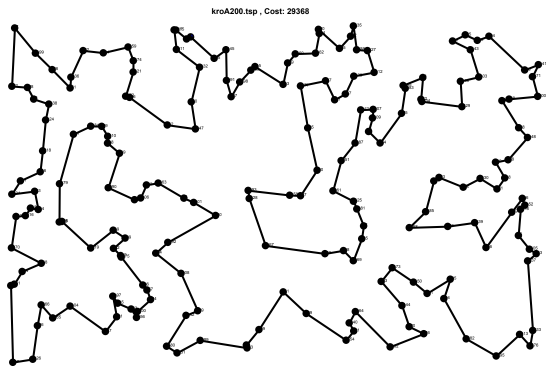
(g)



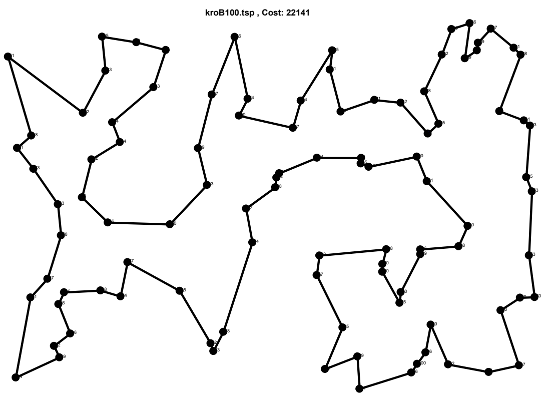
(h)


kroA150.tsp , Cost: 26614

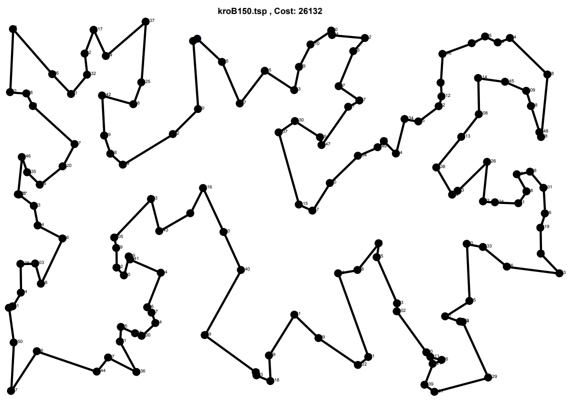
(i)


kroA200.tsp , Cost: 29368

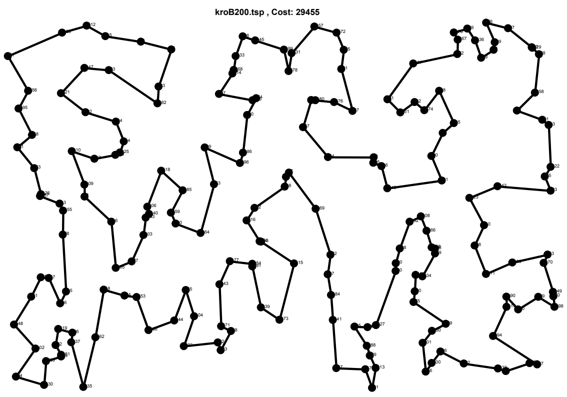
(j)


kroB100.tsp , Cost: 22141

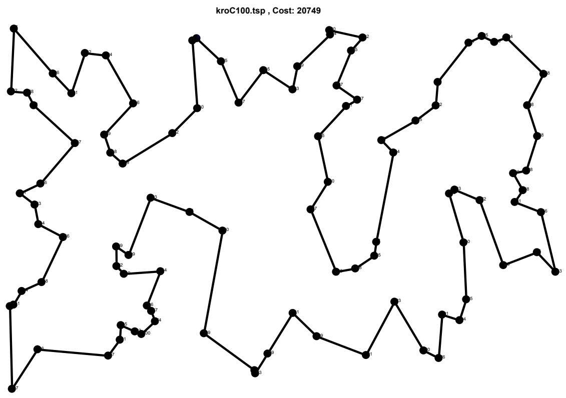
(k)


kroB150.tsp , Cost: 26132

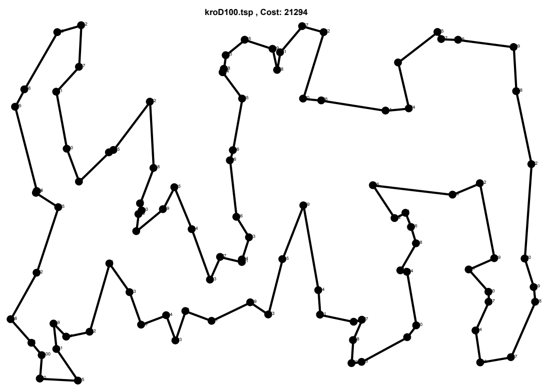
(l)


kroB200.tsp , Cost: 29455

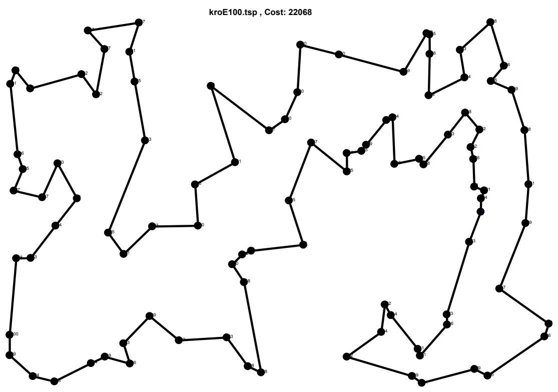
(m)


kroC100.tsp , Cost: 20749

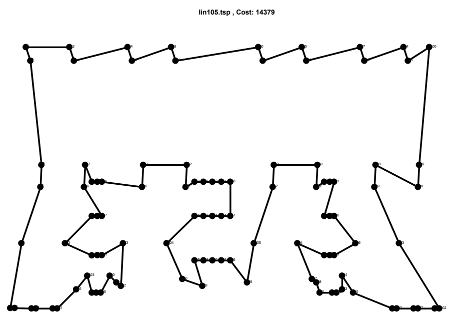
(n)


kroD100.tsp , Cost: 21294

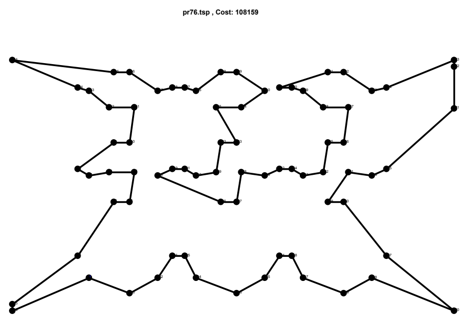
(o)


kroE100.tsp , Cost: 22068

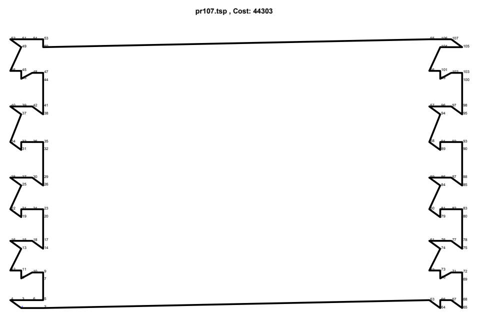
(p)



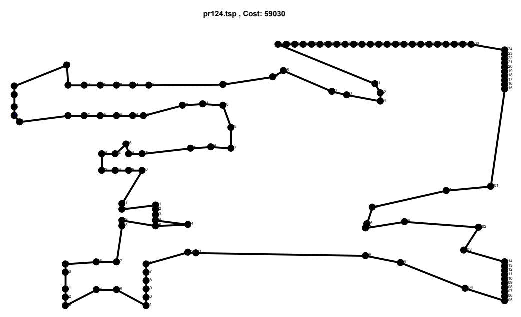
(q)



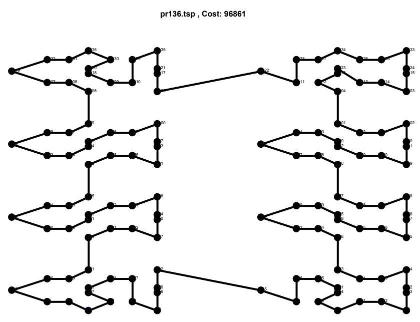
(r)



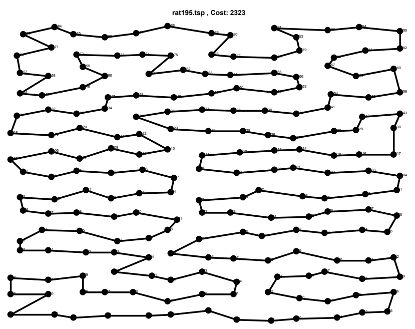
(s)



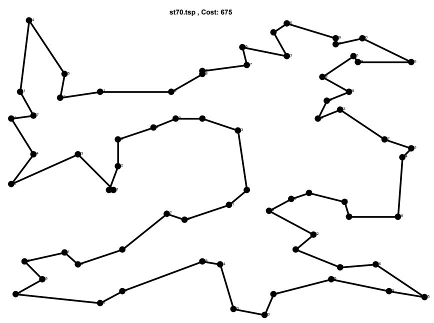
(t)



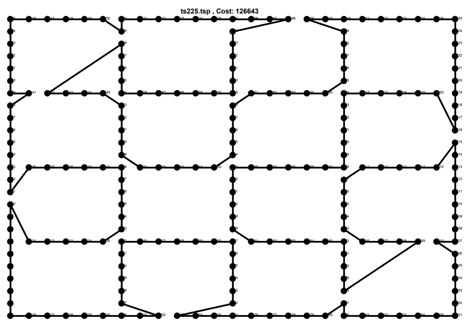
(u)



(v)



(w)



(x)

Figure S1. HCA outputs on benchmark instances. (a) berlin52, Cost = 7542; (b) ch130, Cost = 6110; (c) ch150, Cost = 6528; (d) d198, Cost = 15780; (e) eil51, Cost = 426; (f) eil76, Cost = 538; (g) eil101, Cost = 629; (h) kroA100, Cost = 21282; (i) kroA150, Cost = 26614; (j) kroA200, Cost = 29368; (k) kroB100, Cost = 22141; (l) kroB150, Cost = 26132; (m) kroB200, Cost = 29455; (n) kroC100, Cost = 20749; (o) kroD100, Cost = 21294; (p) kroE100, Cost = 22068; (q) lin105, Cost = 14379; (r) pr76, Cost = 108159; (s) pr107, Cost = 44303; (t) pr124, Cost = 59030; (u) pr136, Cost = 96861; (v) rat195, Cost = 2323; (w) st70, Cost = 675; (x) ts225, Cost = 126643.