REPORT

ON

STUDY OF ALGORITHMS RELATED TO TRAVELLING SALESMAN PROBLEM

By-Kanwardeep Singh (2012C6PS643P)

Events which lead to Travelling Salesman Problem [1] [2] -

- Euler studied the Knight's tour problem around 1750s and 1760s
- Kirkman gave this statement in his book in 1856

  "Given the graph of a polyhedron, can one always final a circuit which passes through each vertex one and only once"
- Icosian game by Hamilton in 1856 – Find paths and circuits on the dodecahedral graph satisfying certain conditions
- Karl Mengar 1930 stated a messenger problem – finding the shortest path joining all of a finite set of points whose pairwise distances are known to the problem solver in euclidean distance.
- Book published in 1931-32 in German talking about TSP
- In the 1950s and 1960s TSP was divided into two categories – Symmetric TSP and Asymmetric TSP

In laymans language TSP is defined as follows – " Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city exactly once and returns to the origin city "

In mathematical terms symmetric TSP is defined as follows :

Let $V = \{a, b, c, \dots., z\}$ be a set of cities, $A = \{(r, s): r, s \in V\}$ be the edge set, and $\delta(r, s) = \delta(s, r)$ be a cost measure associated with edge $(r, s) \in A$

The TSP is the problem of finding a minimal cost closed tour that visits each city once.

In the case cities $r \in V$ are given be their coordinates and δ is the Euclidean distance between r and s , then we have an Euclidean TSP.

Asymmetric TSP same as Symmetric TSP only difference being that

$$\delta(r,s) \neq \delta(s,r)$$

TSP is an NP-hard (Non Deterministic Polynomial-time Hard) problem in combinatorial optimization

Combinatorial optimization means finding an optimal solution from a finite set of solutions.

There are various Algorithms for solving TSP

- Ant System [5]
- Ant Colony System [4]
- Genetic Algorithms[6]
- Simulated Annealing
- Cuckoo Search
- Discrete Cuckoo Search

Ant System

Ant System is based on how the ants behave in real life. In real life ants choose path between 2 or more paths based on probability and pheromone level on the path. Pheromone is a chemical substance used by ants to communicate. Consider Fig.1(a): ants arrive at a decision point in which they have to decide whether to go left or right. Since they don't know which is the best choice, they will choose randomly. It can be expected that, on average, half of the ants decide to turn left and the other half to turn right. This happens both to ants moving from left to right (those whose name begins with an L) and to those moving from right to left (name begins with an R). Fig. 1(b) and (c) shows what happens in the immediately following instants, supposing that all ants walk at approximately the same speed. The number of dashed lines is roughly proportional to the amount of pheromone that the ants have deposited on the ground.

Since the lower path is shorter than the upper one, more ants will visit it on average, and therefore more pheromone will be accumulated. After a short transitory period the difference in the amount of pheromone on the two paths is sufficiently large so as to influence the decision of new ants coming into the system [this is shown by Fig. 1(d)]. From now on, new ants will prefer in probability to choose the lower path, since at the decision point they will perceive a greater amount of pheromone on the lower path. This in turn increases, with a positive feedback due to more ants choosing the shorter path. Very soon all the ants will be using the shorter path.

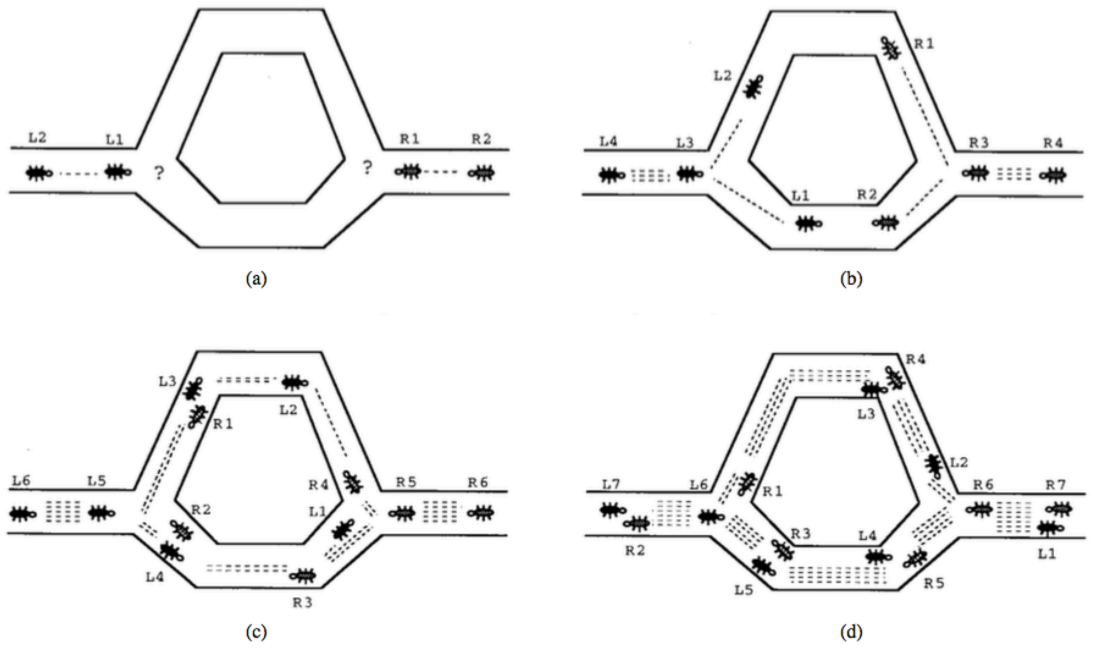


Fig. 1. How real ants find a shortest path. (a) Ants arrive at a decision point. (b) Some ants choose the upper path and some the lower path. The choice is random.(c) Since ants move at approximately a constant speed , the ants which choose the lower, shorter,path reach the opposite decision pint faster than those which choose the upper ,onger,path. (d) Pheromone accumulates at a higher rate on the shorter path. The number of dashed lines is apporximately proportional to the amount of pheromone deposited by ants.

A set of agents known as *ants* search in parallel for a good solution to TSP and communicate with each other via pheromone – mediated indirect and global communication. Each ant constructs its own solution in an iterative manner. Every ant adds new cities to a partial solution by exploiting both information gained from past experience and a greedy heuristic. Memory exists in the form of pheromone deposited on the edges while heuristic information is simply given by the edge's length.

Each edge (r , s) also has a desirability measure called pheromone denoted by $\tau(r,s)$, which is updated at runtime by ants. Each ant generates its own solution by choosing the cities according to a probabilistic *state transition rule.* Ants prefer to move to cities which are connected by short edges and which have high amount of pheromone.

Once all ants have completed the tour a *global pheromone updating rule* is applied i.e a fraction of the pheromone evaporates on all edges and then each ant deposits an amount of pheromone on edges which belong to its tour in proportion to how short its tour was. The process is then iterated. The state transition rule used by ant system, called random-proportional rule is given by (1) which gives the probability with which ant k in city r chooses to move to the city s

$$p_k(r,s) = \begin{cases} \frac{[\tau(r,s)]\cdot[\eta(r,s)]^\beta}{\sum_{u \in J_k(r)}[\tau(r,u)]\cdot[\eta(r,u)]^\beta}, & if\ s \in J_k(r) \\ 0, \ Otherwise \end{cases} \qquad (1)$$

where τ is the pheromone, $\eta = \frac{1}{\delta}$ is the inverse of the distance $\delta(s,r)$, $J_k(s,r)$ is the set of cities that remain to be visited by ant k positioned on city r and β is a parameter which determines the relative importance of pheromone versus distance $(\beta > 0)$.

In (1) we multiply the pheromone on edge (r , s) by the corresponding heuristic value $\eta(r,s)$ In this way we favor the choice of edges which are shorter and which have a greater amount of pheromone.

In ant system, the global updating rule is implemented as follows. Once all ants have built their tours, pheromone is updated on all edges according to

$$\tau(r,s) \leftarrow (1-\alpha).\tau(r,s) + \sum_{k=1}^{m} \Delta\tau_k\ (r,s) \quad (2)$$

where

$$\Delta\tau_k\,(r,s) = \begin{cases} \dfrac{1}{L_k}, & if\ (r,s) \in tour\ done\ by\ ant\ k \\ 0, & otherwise \end{cases}$$

$0 < \alpha < 1$ is a pheromone decay parameter, $L_k$is the length of the tour performed by ant k, and m is the number of ants.

Pheromone updating is used to give more importance to shorter tours. The pheromone updating formula was used because it changes the amount of pheromone in 2 ways. First decreases by evaporation and second more pheromone deposition by the ants.

Although ant system is useful for discovering good solutions for small TSP (upto 30 cities) the time required to find such results for larger number of cities is quite large so it cant scaled with increaseing number of cities. ACS overcomes the problems faced by ant system.

Ant Colony System (ACS)

The ACS is different from Ant System in these 3 ways:
1-The state transition rule provides a direct way to balance between exploration of new edges and exploitation of a priori and accumulated knowledge about the problem.
2-The global updating rule is applied only to edges which belong to the best ant tour.
3-While ants construct a solution a local pheromone updating rule is applied.

ACS works as follows: M ants are initially positioned on n cities chosed randomly. Each ant builds a tour be repeatedly applying a stochastic greedy rule (the state transition rule). While constructing its tour, an ant also modifies the amount of pheromone on the visited edges by applying the local updating rule. Once all ants have terminated their tour . The amount of pheromone on edges is modified by applying the global updating rule. As was the case in ant system, ants are guided in building their tours, by both heurristic information and by pheromone information. An edge with high amount of pheromone is very desirable choice. The pheromone updating rules are designed so that they tend to give more pheromone to edges which should be visited by ants. The ACS algorithm is reported in Fig. 3. The 3 rules are discussed below.

```
Initialize
**Loop/* at this level each loop is called an iteration*/**
    Each ant is positioned on a starting node
    **Loop /* at this level each loop is called a step */**
        Each ant applies a state transition rule to incrementally build a
        solution and a local pheromone updating rule
    **Until** all ants have built a complete solution
    A global pheromone updating rule is applied
**Until** End_condition
```

Fig. 3. The ACS algorithm

A. ACS State Transition Rule

In the ACS the state transition rule is as follows: an ant positioned on node r chooses the city s to move to by applying the rule given by (3)

$$s = \begin{cases} \arg max_{u \in J_k(r)} \{[\tau(r,u)].[\eta(r,u)]^{\beta}\}, & if \;\; q \leq q_0 \;\; (exploitation) \\ S, & otherwise \; (biased \; exploration) \end{cases} \quad (3)$$

where q is a random number uniformly distrubuted in [0…1], q_0 is a parameter (0 ≤ q_0≤ 1), and S is a random variable selected according to the probability distribution given in (1).

The state transition rule resulting from (3) and (1) is called pseudo-random-proportional rule. This state transition rule, as with the previous random-proportional rule , favors transitions toward nodes connected by short edges and with a large amount of pheromone. The parameter q_0 determines the relative improtance of exploitation versus exploration: every time an ant in city r has to choose a city s to move to, it samples a random number 0 ≤ q ≤ 1. If q ≤ q_0 then the best edge, according to (3), is chosen (exploitation), otherwise an edge is chosen according to (1) (biased exploration).

B. ACS Global Updating Rule

In ACS only the globally best ant is allowed to deposit pheromone. This choice, together with the use of the pseudo-random-proportional rule, is intended to make the search more directed. Global updating is performed after all ants have completed their tours. The pheromone level is updated by applying the global updating rule of (4)

$$\tau(r,s) \leftarrow (1-\alpha).\tau(r,s) + \alpha.\Delta\tau(r,s) \qquad (4)$$

where

$$\Delta\tau(r,s) = \begin{cases} (L_{gb})^{-1}, & if(r,s) \in global-best-tour \\ 0, & otherwise \end{cases}$$

0 < α < 1 is the pheromone decay parameter, and L_{gb} is the length of the globally best tour from the beginning of the trial. The global updating rule is intended to provide greater amount of pheromone to shorter tours. Equation (4) dictates that only those edges belonging to the globally best tour will receive reinforcement. The paper also tested another type of global updating rule called *iteration-best* which instead used L_{ib} (the length of the best tour in the current iteration of the trial) in (4). Experiments have shown that the difference between the 2 schemes is not much with a slight preference to global-best. For the following experiments we would be using global-best.

C. ACS Local Updating Rule

While building a solution (i.e a tour) of the TSP ants visit edges and change their pheromone level by applying the local updating rule of (5)

$$\tau(r,s) \leftarrow (1-\rho).\tau(r,s) + \rho.\Delta\tau(r,s) \qquad (5)$$

where 0 < ρ < 1 is a parameter.

We can choose any value of $\Delta\tau(r,s)$. We can take it as 0 or $\tau_0$or $\gamma.max_{z\in J_k(s)}$. Results show that taking its value as 0 gives the worst result and taking it as τ_0 or $\gamma.max_{z\in J_k(s)}$ gives almost the same results with a slighty better results with $\gamma.max_{z\in J_k(s)}$, but because it is a bit computation intensive we would be using $\Delta\tau(r,s) = \tau_0$ for the rest of our discussion.

D. ACS Parameter Setttings

In all experiments of the following sections the numeric parameters, except when indicated differently, are set to the following values: β = 2, q_0= 0.9, α = ρ = 0.1 , τ_0 = $(n.L_{nn})^{-1}$where L_{nn} is the tour length produced by the nearest neighbor heuristic [3] and n is the number of cities. These values were obtained by a preliminary optimization phase, in which we found that the experimental optimal values of the parameters were largely independent of the problem, except for τ_0 for which, as we said, τ_0 = $(n.L_{nn})^{-1}$The number of ants used is m = 10, which is the optimal number of ants determined experimentally.

E. ACS Importance of the pheromone and Heuristic Function

Experimental results have shown that the heuristic function η is fundamental in making the algorithm find good solutions in a reasonable time.In fact , when β = 0 ACS performance worsens significantly.
Using no pheromone also deteriarates performance.
The reason that ACS without the heuristic function performs better than ACS without pheromone is that in the first case, although not helped by heuristic information, the ACS is still guided by reinforcement provided by the global updating rule in the form of pheromone, while in the second case the ACS reduces to a multigreedy algorithm.

F.Comparison with other Heuristics

Table III reports the results on the geometric instances. The heuristics with which we compare the ACS in this case are a genetic algorithm (GA), evolutionary programming (EP), and simulated annealing (SA). The ACS is run for 1250 iterations using 20 ants (this amounts to approximately the same number of tours searched by the heuristics with which we compare our results). ACS results are averaged over 15 trials. In this case comparison is performed on the best results, as opposed to average results as in previous Table II (this choice was dictated by the availability of published results). The difference between integer and real tour length is that in the first case distances between cities are measured by integer numbers, while in the second case by floating point approximations of real numbers. Results using EP are from [15], and those using GA are from [41] for Eil50 and Eil75, and from [6] for KroA100. Results using SA are from [29]. Eil50, Eil75 are from [14] and are included in TSPLIB with an additional city as Eil51.tsp and Eil76.tsp.

KroA100 is also in TSPLIB. The best result for each problem is in boldface. Again, the ACS offers the best performance in nearly every case. Only for the Eil50 problem does it find a slightly worse solution using real-valued distance as compared with EP, but the ACS only visits 1830 tours, while EP used 100 000 such evaluations (although it is possible that EP found its best solution earlier in the run, this is not specified in the paper [15]).

TABLE III

COMPARISON OF ACS WITH OTHER HEURISTICS ON GEOMETRIC INSTANCES OF THE SYMMETRIC TSP. WE REPORT THE BEST INTEGER TOUR LENGTH, THE BEST REAL TOUR LENGTH (IN PARENTHESES), AND THE NUMBER OF TOURS REQUIRED TO FIND THE BEST INTEGER TOUR LENGTH (IN SQUARE BRACKETS). N/A MEANS "NOT AVAILABLE." IN THE LAST COLUMN THE OPTIMAL LENGTH IS AVAILABLE ONLY FOR INTEGER TOUR LENGTHS

Problem name	ACS	GA	SA	Optimum
Eil50	**425**	428	443	**425**
(50-city problem)	(427.96)	(N/A)	(N/A)	(N/A)
	[1,830]	[25,000]	[68,512]	
Eil75	**535**	545	580	**535**
(75-city problem)	**(542.37)**	(N/A)	(N/A)	(N/A)
	[3,480]	[80,000]	[173,250]	
KroA100	**21,282**	21,761	N/A	**21,282**
(100-city problem)	**(21,285.44)**	(N/A)	(N/A)	(N/A)
	[4,820]	[103,000]	[N/A]	

G.Future Improvements

There are a number of ways in which the ant colony approach can be improved and/or changed. A first possibility regards the number of ants which should contribute to the global updating rule. In ant system all the ants deposited their pheromone, while in the ACS only the best one does: obviously there are intermediate possibilities.

Another change to the ACS could be, again taking inspiration from [1], allowing ants which produce very bad tours to subtract pheromone.

A second possibility is to move from the current parallel local updating of pheromone to a sequential one. In the ACS all ants apply the local updating rule in parallel, while they are building their tours. We could imagine a modified ACS in which ants build tours sequentially: the first ant starts, builds its tour, and as a side effect, changes the pheromone on visited edges. Then the second ant starts and so on until the last of the $m$ants has built its tour. At this point the global updating rule is applied. This scheme will determine a different search regime, in which the preferred tour will tend to remain the same for all the ants (as opposed to the situation in the ACS, in which local updating shuffles the tours). Nevertheless, the search will be diversified since the first ants in the sequence will search in a neighborhood of the preferred tour that is more narrow than later ones.

Genetic Algorithms

A simple genetic algorithm can be defined in the following steps:

Step 1- Create an initial population of P chromosomes (generation 0)

Step 2- Evaluate the fitness of each chromosome

Step 3- Select P parents from the current population via proportional selection (ie the selection probability is proportional to the fitness)

Step 4- Choose at random a pair of parents for mating . Exchange bit strings with the one point crossover to create 2 offspring

Step 5- Process each offspring by a mutation operator and insert the resulting offspring in the new population

Step 6-Repeat steps 4 and 5 until all parents are selected and mated (P offspring are created)

Step 7-Replace old population of chromosomes in the new population by the new one.

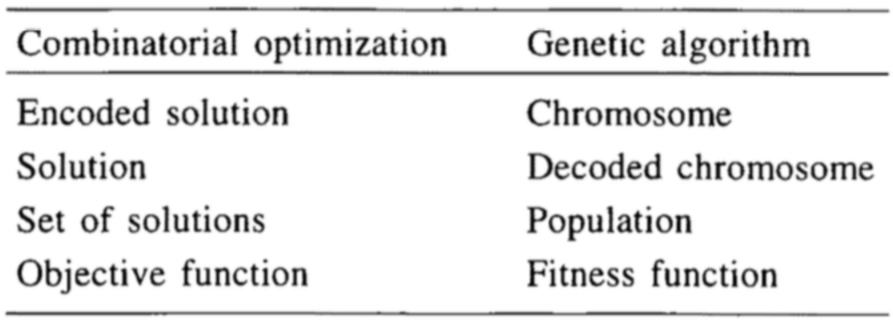
Step 8-Evaluate the fitness of each chromosome in the new population.

Step 9- Go back to step 3 if the number of generations is less than some upper bound . Otherwise the final result is the best chromosome created during the search.

Underlying principles

A genetic algorithm operates on a finite population of chromosomes or bit strings . The search mechanism consists of 3 different phases :

Evaluation of the fitness of each chromosome , selection of the parent chromosomes and application of the mutation and recombination operators to the parent chromosomes. The new chromosomes resulting from these operations form the next generation and the process is repeated until the system ceases to improve.



Combinatorial optimization	Genetic algorithm
Encoded solution	Chromosome
Solution	Decoded chromosome
Set of solutions	Population
Objective function	Fitness function

In TSP each chromosome encodes a solution to the problem (ie a tour). The fitness of chromosome is related to the tour length which in turn depends on the ordering of the cities. Since the TSP is a minimization problem the tour lengths must be transformed so that high fitness values are associated with short tours. This can be done by subtracting each tour length from the maximum tour length found in the current population.

The genetic algorithm searches the space of solutions by combining the best features of two good tours into a single one. Since the fitness is related to the length of the edges included in the tour, it is clear that the edges represent the basic information to be transferred to the offspring.

The success or failure of the approaches described in the following sections can often be explained by their ability or inability to adequately represent and combine the edge information in the offspring.

The ordinal representation

This technique can be used to form one point crossover which will always generate feasible offspring . This is not very good representation because the sequencing information in the two parent chromosomes is not transferred to the offspring and resulting search is close to a random search

For example let the tour be 12564387 for 8 city problem ie a person goes from 1 to 2 to 5 and so on. The encoding is done in the following way. The position of the current city in the canonic tour is stored at the corresponding position in the resulting chromosome. The tour is updated by deleting that city and the procedure is repeated.

Current tour	Canonic tour	Ordinal representation
$\underline{1}$ 2 5 6 4 3 8 7	$\underline{1}$ 2 3 4 5 6 7 8	1
1 $\underline{2}$ 5 6 4 3 8 7	$\underline{2}$ 3 4 5 6 7 8	1 1
1 2 $\underline{5}$ 6 4 3 8 7	3 4 $\underline{5}$ 6 7 8	1 1 3
1 2 5 $\underline{6}$ 4 3 8 7	3 4 $\underline{6}$ 7 8	1 1 3 3
1 2 5 6 $\underline{4}$ 3 8 7	3 $\underline{4}$ 7 8	1 1 3 3 2
1 2 5 6 4 $\underline{3}$ 8 7	$\underline{3}$ 7 8	1 1 3 3 2 1
1 2 5 6 4 3 $\underline{8}$ 7	7 $\underline{8}$	1 1 3 3 2 1 2
1 2 5 6 4 3 8 $\underline{7}$	$\underline{7}$	1 1 3 3 2 1 2 1

Figure 6. The ordinal representation.

The resulting chromosome can be easily decoded into original be the inverse process . Interestingly 2 parent chromosomes encoded in this way always generate a feasible offspring when they are processed by the one point crossover.

The path representation

The path representation is a natural way to encode TSP tours. The crossover operators based on this representation typically generate offspring that inherit the relative order or the absolute position of the cities from the parent chromosomes.

Crossover Operators preserving the absolute position

1-Partially mapped crossover(PMX)

This operator first randomly selects two cut points on both parents. In order to create an offspring, the substring between the two cut points in the first parent replaces the corresponding substring in the second parent. Then, the inverse replacement is applied outside of the cut points, in order to eliminate duplicates and recover all cities.

In figure 8, the offspring is created by first replacing the substring 236 in parent 2 by the substring 564. Hence, city 5 replaces city 2, city 6 replaces city 3, and city 4 replaces city 6 (step 1). Since cities 4 and 5 are now duplicated in the offspring, the inverse replacement is applied outside of the cut points. Namely, city 2 replaces city 5, and city 3 replaces city 4 (step 2). In the latter case, city 6 first replaces city 4, but since city 6 is already found in the offspring at position 4, city 3 finally replaces city 6. Multiple replacements at a given position occur when a city is located between the cut points on both parents, like city 6 in this example.

parent 1	:	**1**	**2**	|	**5**	**6**	**4**	|	**3**	**8**	**7**
parent 2	:	1	4	|	2	3	6	|	5	7	8
offspring											
(step 1)	:	1	4		**5**	**6**	**4**		5	7	8
(step 2)	:	1	3		**5**	**6**	**4**		2	7	8

Figure 8. The partially-mapped crossover.

2-Cycle crossover (CX)

The cycle crossover focuses on subsets of cities that occupy the same subset of positions in both parents. Then, these cities are copied from the first parent to the offspring (at the same positions), and the remaining positions are filled with the cities of the second parent. In this way, the position of each city is inherited from one of the two parents. However, many edges can be broken in the process, because the initial subset of cities is not necessarily located at consecutive positions in the parent tours. In figure 9, the subset of cities {3, 4, 6 } occupies the subset of positions {2, 4, 5 } in both parents. Hence, an offspring is created by filling the positions 2, 4 and 5 with the cities found in parent 1, and by filling the remaining positions with the cities found in parent 2. Note that the crossover operator introduced in [6] is a restricted version of CX, where the subset of cities must occupy consecutive positions in both parents.

parent 1	:	**1**	**3**	**5**	**6**	**4**	**2**	**8**	**7**
parent 2	:	1	4	2	3	6	5	7	8
offspring	:	1	**3**	2	**6**	**4**	5	7	8

Figure 9. The cycle crossover.

Crossover operators preserving the relative order

1-Modified Crossover

This crossover operator is an extension of the one-point crossover for permutation problems. A cut position is chosen at random on the first parent chromosome. Then, an offspring is created by appending the second parent chromosome to the initial segment of the first parent (before the cut point), and by eliminating the duplicates. An example is provided in figure 10.

Note that the cities occupying the first positions in parent 2 tend to move forward in the resulting offspring (with respect to their positions in parent 1). For example, city 4 occupies the fifth position in parent 1, but since it occupies the second position in parent 2, it moves to the third position in the resulting offspring

parent 1	:	**1**	**2**	|	**5**	**6**	**4**	**3**	**8**	**7**
parent 2	:	1	4		2	3	6	5	7	8
offspring	:	**1**	**2**		4	3	6	5	7	8

Figure 10. The modified crossover.

2-Order Crossover (OX)

This crossover operator extends the modified crossover of Davis by allowing two cut points to be randomly chosen on the parent chromosomes. In order to create an offspring, the string between the two cut points in the first parent is first copied to the offspring. Then, the remaining positions are filled by considering the sequence of cities in the second parent, starting after the second cut point (when the end of the chromosome is reached, the sequence continues at position 1).

In figure 11, the substring 564 in parent 1 is first copied to the offspring (step 1). Then, the remaining positions are filled one by one after the second cut point, by considering the corresponding sequence of cities in parent 2, namely 57814236 (step 2). Hence, city 5 is first considered to occupy position 6, but it is discarded because it is already included in the offspring. City 7 is the next city to be considered, and it is inserted at position 6. Then, city 8 is inserted at position 7, city 1 is inserted at position 8, city 4 is discarded, city 2 is inserted at position 1, city 3 is inserted at position 2, and city 6 is discarded. Clearly, OX tries to preserve the relative order of the cities in parent 2, rather than their absolute position. In figure 11, the offspring does not preserve the position of any city in parent 2. However, city 7 still appears before city 8 and city 2 before city 3 in the resulting offspring.

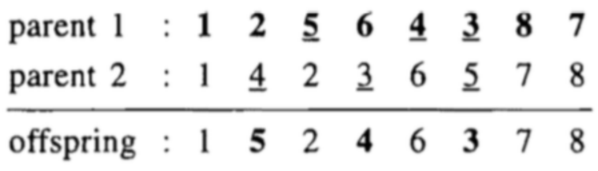
parent 1	:	**1**	**2**	|	**5**	**6**	**4**	|	**3**	**8**	**7**
parent 2	:	1	4	|	2	3	6	|	5	7	8
offspring											
(step 1)	:	–	–		**5**	**6**	**4**		–	–	–
(step 2)	:	2	3		**5**	**6**	**4**		7	8	1

Figure 11. The order crossover.

3-Order Based Crossover (OBX)

This crossover also focuses on the relative order of the cities on the parent chromosomes. First, a subset of cities is selected in the first parent. In the offspring, these cities appear in the same order as in the first parent, but at positions taken from the second parent. Then, the remaining positions are filled with the cities of the second parent.

In figure 12, cities 5, 4 and 3 are first selected in parent I, and must appear in this order in the offspring. Actually, these cities occupy positions 2, 4 and 6 in parent 2. Hence, cities 5, 4 and 3 occupy positions 2, 4 and 6, respectively, in the offspring. The remaining positions are filled with the cities found in parent 2.

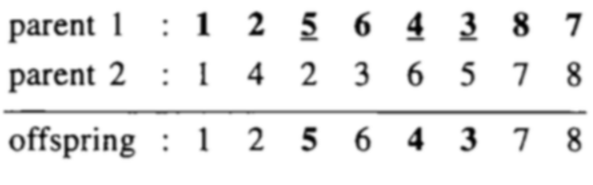


parent 1	:	**1**	**2**	$\underline{\mathbf{5}}$	**6**	$\underline{\mathbf{4}}$	$\underline{\mathbf{3}}$	**8**	**7**
parent 2	:	1	$\underline{4}$	2	$\underline{3}$	6	$\underline{5}$	7	8
offspring	:	1	**5**	2	**4**	6	**3**	7	8

Figure 12. The order-based crossover.

4-Position Based Crossover (PBX)

Here, a subset of positions is selected in the first parent. Then, the cities found at these positions are copied to the offspring (at the same positions). The other positions are filled with the remaining cities, in the same order as in the second parent. The name of this operator is a little bit misleading, because it is the relative order of the cities that is inherited from the parents (the absolute position of the cities inherited from the second parent is rarely preserved). This operator can be seen as an extension of the order crossover OX, where the cities inherited from the first parent do not necessarily occupy consecutive positions. In figure 13, positions 3, 5 and 6 are first selected in parent 1. Cities 5, 4 and 3 are found at these positions, and occupy the same positions in the offspring. The other positions are filled one by one, starting at position 1, by inserting the remaining cities according to their relative order in parent 2, namely 12678.



parent 1	:	**1**	**2**	$\underline{\mathbf{5}}$	**6**	$\underline{\mathbf{4}}$	$\underline{\mathbf{3}}$	**8**	**7**
parent 2	:	1	4	2	3	6	5	7	8
offspring	:	1	2	**5**	6	**4**	**3**	7	8

Figure 13. The position-based crossover.

The adjacency representation

The adjacency representation is designed to facilitate the manipulation of edges. The crossover operators based on this representation generate offspring that inherit most of their edges from the parent chromosomes.

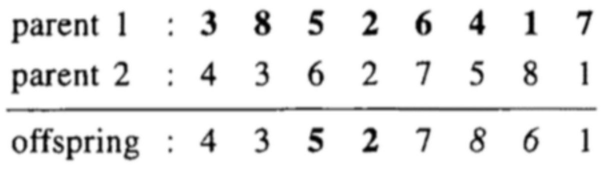
The adjacency representation can be described as follows: city j occupies position i in the chromosome if there is an edge from city i to city j in the tour. For example, the chromosome 38526417 encodes the tour 13564287. City 3 occupies position 1 in the chromosome because edge (1, 3) is in the tour. Similarly, city 8 occupies position 2 because edge (2, 8) is in the tour, etc. As opposed to the path representation, a tour has only two different adjacency representations for symmetric TSPs.

1-*Alternate Edges Crossover*

This operator is mainly of historical interest. The results with the operator have been uniformly discouraging. However, it is a good introduction to the other edge-preserving operators.

Here, a starting edge *(i,j)* is selected at random in one parent. Then, the tour is extended by selecting the edge (j, k) in the other parent. The tour is progressively extended in this way by alternatively selecting edges from the two parents. When an edge introduces a cycle, the new edge is selected at random (and is not inherited from the parents).

In figure 14, an offspring is generated from two parent chromosomes that encode the tours 13564287 and 14236578, respectively, using the adjacency representation. Here, edge (1, 4) is first selected in parent 2, and city 4 in position I of parent 2 is copied at the same position in the offspring.



parent 1	:	**3**	**8**	**5**	**2**	**6**	**4**	**1**	**7**
parent 2	:	4	3	6	2	7	5	8	1
offspring	:	4	3	**5**	**2**	7	*8*	*6*	1

Figure 14. The alternate edges crossover.

Then, the edges (4, 2) in parent 1, (2, 3) in parent 2, (3, 5) in parent 1 and (5, 7) in parent 2 are selected and inserted in the offspring. Then, edge (7, 1) is selected in parent 1. However, this edge introduces a cycle, and a new edge leading to a city not yet visited is selected at random. Let us assume that (7, 6) is chosen. Then, edge (6, 5) is selected in parent 2, but it also introduces a cycle. At this point, (6, 8) is the only selection that does not introduce a cycle. Finally, the tour is completed with edge (8, 1).

The final offspring encodes the tour 14235768, and all edges in the offspring are inherited from the parents, apart from the edges (7, 6) and (6, 8). In the above description, an implicit orientation of the parent tours is assumed. For symmetric problems, the two edges that are incident to a given city can be considered. In the above example, when we get to city 7 and select the next edge in parent 1, edges (7, 1) and (7, 8) can both be considered. Since (7, 1) introduces a cycle, edge (7, 8) is selected. Finally, edges (8, 6) and (6, 1) complete the tour.

parent 1	:	**3**	**8**	**5**	**2**	**6**	**4**	**1**	**7**
parent 2	:	4	3	6	2	7	5	8	1
offspring	:	4	3	**5**	**2**	7	1	8	6

Figure 15. The alternate edges crossover (revisited).

2-*Edge Recombination crossover*(ER)

Quite often, the alternate edge operator introduces many random edges in the offspring, particularly the last edges, when the choices for extending the tour are limited. Since the offspring must inherit as many edges as possible from the parents, the introduction of random edges should be minimized. The edge recombination operator reduces the myopic behavior of the alternate edge approach with a special data structure called the "edge map".

Basically, the edge map maintains the list of edges that are incident to each city in the parent tours and that lead to cities not yet included in the offspring. Hence, these edges are still available for extending the tour and are said to be active. The strategy is to extend the tour by selecting the edge that leads to the city with the minimum number of active edges. In the case of equality between two or more cities, one of these cities is selected at random. With this strategy, the approach is less likely to get trapped in a "dead end", namely, a city with no remaining active edges (thus requiring the selection of a random edge).

For the tours 13564287 and 14236578 (path representation), the initial edge map is shown in figure 16.

The operation of the edge recombination crossover operator will now be illustrated on this initial edge map (see figure 17). To this end, the edge map will be updated after each city selection. In these edge maps, cities of particular interest are underlined (namely, cities that are adjacent to the selected city in the parents and are not visited yet).

Let us assume that city 1 is selected as the starting city. Accordingly, all edges incident to city 1 must first be deleted from the initial edge map. From city 1, we can go to cities 3, 4, 7 or 8. City 3 has three active edges, while cities 4, 7 and 8 have two active edges, as shown by edge map (a) in figure 17. Hence, a random choice is made between cities 4, 7 and 8. We assume that city 8 is selected. From 8, we can go to cities 2 and 7. As indicated in edge map (b), city 2 has two active edges and city 7 only one, so the latter is selected. From city 7, there is no choice but to go to city 5. From this point, edge map (d) offers a choice between cities 3 and 6 with two active edges. Let us assume that city 6 is randomly selected. From city 6, we can go to cities 3 and 4, and edge map (e) indicates that both cities have one active edge. We assume that city 4 is randomly selected. Finally, from city 4 we can only go to city 2, and from city 2 we must go to city 3. The final tour is 18756423 and all edges are inherited from both parents.

city 1 has edges to	:	3 4 7 8
city 2 has edges to	:	3 4 8
city 3 has edges to	:	1 2 5 6
city 4 has edges to	:	1 2 6
city 5 has edges to	:	3 6 7
city 6 has edges to	:	3 4 5
city 7 has edges to	:	1 5 8
city 8 has edges to	:	1 2 7

Figure 16. The edge map.

City 1 is selected

(a)

city 2 has edges to	:	3 4 8
city $\underline{3}$ has edges to	:	$\underline{2}$ $\underline{5}$ $\underline{6}$
city $\underline{4}$ has edges to	:	$\underline{2}$ $\underline{6}$
city 5 has edges to	:	3 6 7
city 6 has edges to	:	3 4 5
city $\underline{7}$ has edges to	:	$\underline{5}$ $\underline{8}$
city $\underline{8}$ has edges to	:	$\underline{2}$ $\underline{7}$

City 8 is selected

(b)

city $\underline{2}$ has edges to	:	$\underline{3}$ $\underline{4}$
city 3 has edges to	:	2 5 6
city 4 has edges to	:	2 6
city 5 has edges to	:	3 6 7
city 6 has edges to	:	3 4 5
city $\underline{7}$ has edges to	:	$\underline{5}$

City 7 is selected

(c)

city 2 has edges to	:	3 4
city 3 has edges to	:	2 5 6
city 4 has edges to	:	2 6
city $\underline{5}$ has edges to	:	$\underline{3}$ $\underline{6}$
city 6 has edges to	:	3 4 5

City 5 is selected

(d)

city 2 has edges to	:	3 4
city $\underline{3}$ has edges to	:	$\underline{2}$ $\underline{6}$
city 4 has edges to	:	2 6
city $\underline{6}$ has edges to	:	$\underline{3}$ $\underline{4}$

City 6 is selected

(e)

city 2 has edges to	:	3 4
city $\underline{3}$ has edges to	:	$\underline{2}$
city $\underline{4}$ has edges to	:	$\underline{2}$

City 4 is selected

(f)

city 2 has edges to	:	$\underline{3}$
city 3 has edges to	:	2 6

City 2 is selected

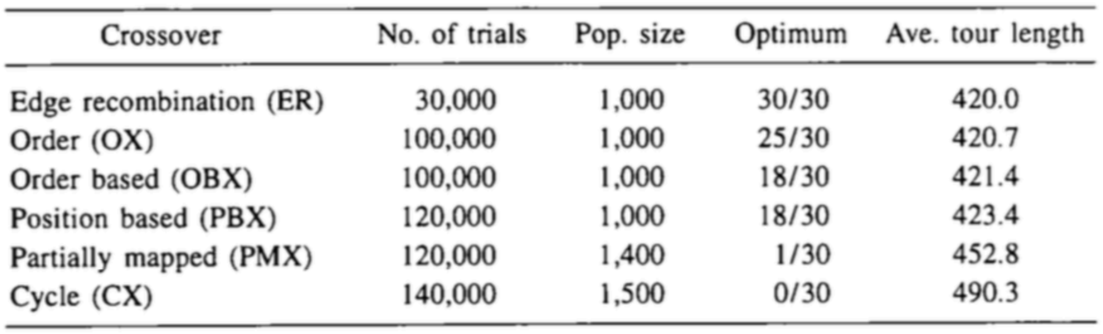
(g)

city $\underline{3}$ has edges to	:	

City 3 is selected

Figure 17. Evolution of the edge map.

Comparison among various techniques discussed above


Comparison of six crossover operators (30 runs) in [55].

Crossover	No. of trials	Pop. size	Optimum	Ave. tour length
Edge recombination (ER)	30,000	1,000	30/30	420.0
Order (OX)	100,000	1,000	25/30	420.7
Order based (OBX)	100,000	1,000	18/30	421.4
Position based (PBX)	120,000	1,000	18/30	423.4
Partially mapped (PMX)	120,000	1,400	1/30	452.8
Cycle (CX)	140,000	1,500	0/30	490.3

Simulated Annealing

The simulated annealing are inspired by how metals cool and form low energy states. In this algorithm we simulate tge annealing process of metal atoms . Metal atoms at a high temperature will become unstable from their initial states and search for other states. While cooling, the metal atoms will find an energy state that is lower than their initial state. The state changing procedure can be applied to solve real-world problems. The system generates a new state and then compares the energy of the new state with the energy of the current state. If the energy of the new state is lower than the energy of the current state, then the system accepts this state. Otherwise, the system changes to this state according to the transition probability P, shown as follows:

$$P = e^{\frac{-\Delta E}{kT}}$$

$$\Delta E = E(S') - E(S)$$

where k is the Boltzmann constant, T is the current temperature of the system, S is the current state, S’ is the new state and E is the energy function. If the system temperature is cooling to a predefined temperature or the maximum number of iterations is met, the system prints out the best state, i.e., the near optimal solution. Fig. 4 shows the flowchart of the simulated annealing algorithm

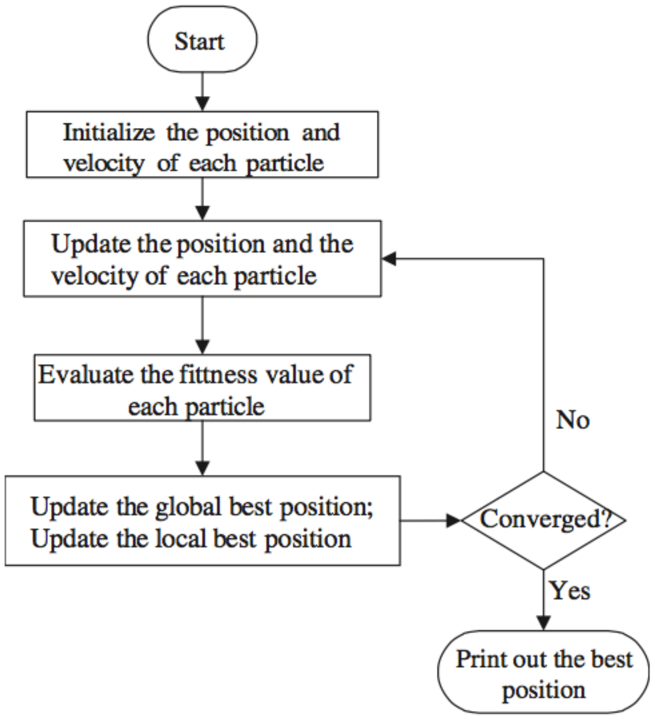

Start
Initialize the position and velocity of each particle
Update the position and the velocity of each particle
Evaluate the fittness value of each particle
Update the global best position; Update the local best position
Converged?
No
Yes
Print out the best position

Fig. 5. The flowchart of the particle swarm optimization (Kennedy & Eberhart, 1995).

Cuckoo Search

Cuckoo search is inspired by cuckoo birds because of their aggressive reproduction stratergy. Quite a number of species engage the obligate brood parasitism by laying their eggs in the nests of other host birds.In addition the timing of egg laying of some species is also amazing.

Levy Flights

Various studies have shown that flight behavior of many animals and insects has demonstrated the typical characteristics of levy flights. A recent study by Reynolds and Frye shows that fruit flies explore their landscape using a series of straight flight paths punctuated by a sudden 90 degree turn leading to a levy flight – style pattern. Studies on human behavior such as Ju Hoansi hunter gatherer foraging patterns show the typical feature of Levy flights.

Cuckoo search algorithm

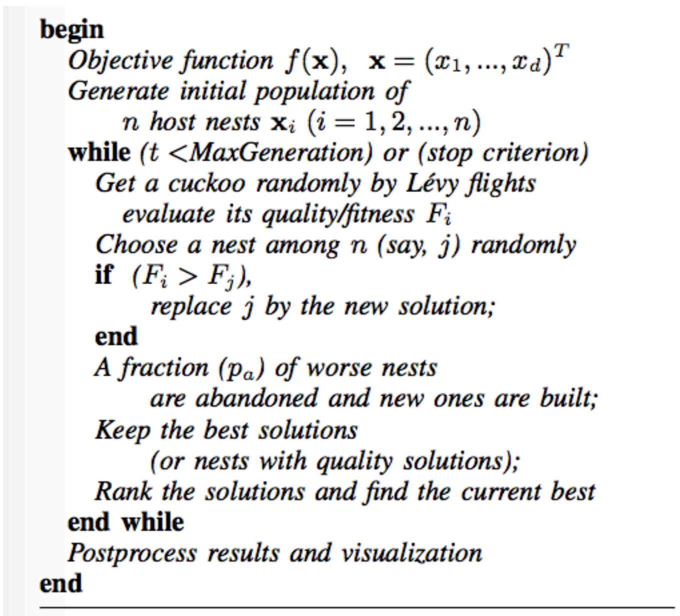
Idealized rules used for cuckoo search algorithm

1-Each cuckoo lays one egg at a time and dump its egg in randomly chosen nest

2-The best nests with high quality of eggs will carry over to the next generation

3-The number of available host nests is fixed and the egg laid by a cuckoo is discovered by the host bird with a probability $p_a \in [0,1]$. In this case the host bird can either throw the egg away or abandon the nest and build a completely new nest.

For simplicity this last assumption can be approximated by the fraction p_a of the n nests are replaced by new nests (with new random solutions). The quality of a solution can simply be proportional to the value of the objective function. We use the following simple representations that each egg in a nest represents a solution and a cuckoo egg represent a new solution, the aim is to use the new and potentially better solutions (cuckoo) to replace a not so good solution in the nests.
Based on these rules the basic steps of the Cuckoo Search (CS) can be summarized as the pseudo code shown in Fig. 1.



```
begin
  Objective function $f(\mathbf{x})$, $\mathbf{x} = (x_1, ..., x_d)^T$
  Generate initial population of
      $n$ host nests $\mathbf{x}_i$ $(i = 1, 2, ..., n)$
  while ($t$ <MaxGeneration) or (stop criterion)
    Get a cuckoo randomly by Lévy flights
      evaluate its quality/fitness $F_i$
    Choose a nest among $n$ (say, $j$) randomly
    if ($F_i > F_j$),
        replace $j$ by the new solution;
    end
    A fraction ($p_a$) of worse nests
        are abandoned and new ones are built;
    Keep the best solutions
        (or nests with quality solutions);
    Rank the solutions and find the current best
  end while
  Postprocess results and visualization
end
```

Fig. 1. Pseudo code of the Cuckoo Search (CS).

When generating new solutions x^{t+1} for say a cuckoo i a levy flight is performed

$$x_i^{t+1} = x_i^{t} + \alpha \oplus \; levy\,(\lambda) \qquad (1)$$

where $\alpha > 0$ is the step size which should be related to the scales of the problem of interests. In most cases we can use $\alpha = 1$. The above equation is essentially the stochastic equation for random walk. In general, a random walk is a Markov chain whose next status/location only depends on the current location (the first term in the above equation) and the transition probability (the second term). The product ⊕ means entry wise multiplications. This entry wise product is similar to those used in PSO, but here the random walk via Levy flight is more efficient in exploring the search space as its step length is much longer in the long run.

The Levy flight essentially provides a random walk while the random step length is drawn from a Levy distribution

$$levy \sim u = t^{-\lambda} \ (1 < \lambda \leq 3)$$

which has an infinite variance with an infinite mean. Here the steps essentially form a random walk process with a power- law step-length distribution with a heavy tail. Some of the new solutions should be generated by Levy walk around the best solution obtained so far, this will speed up the local search. However, a substantial fraction of the new solutions should be generated by far field randomization and whose locations should be far enough from the current best solution, this will make sure the system will not be trapped in a local optimum.

Discrete Cuckoo Algorithm

There is still some room for improvement in CS, both in terms of the inspiration source and of the algorithm itself. The strength of CS is the way how to exploit and explore the solution space by a cuckoo. This cuckoo can have some 'intelligence' so as to find much better solutions. So we can control the intensification and diversification through the cuckoo's mobility in the search space. The proposed improvement considers a cuckoo as a first level of con- trolling intensification and diversification, and since such a cuckoo is an individual of a population, we can qualify the population as a second level of control, which can be restructured by adding a new category of cuckoos smarter and more efficient in their search.

Studies show that cuckoos can also engage a kind of surveillance on nests likely to be a host . This behaviour can serve as an inspiration to create a new category of cuckoos that have the ability to change the host nest during incubation to avoid abandonment of eggs. These cuckoos use mechanisms before and after brooding such as the observation of the host nest to decide if the nest is the best choice or not (so, it looks for a new nest much better for the egg). In this case, we can talk about a kind of local search performed by a fraction of cuckoos around current solutions. For simplicity, we can divide the mechanism adopted by this new fraction of cuckoos in our proposed approach into two main steps: (1) A cuckoo , initially moves by Levy flights towards a new solution (which represents a new area); (2) from the current solution, the cuckoo in the same area seeks a new, better solution. According to these two steps, we can see that the search mechanism with a new fraction p_c can be directly introduced in the standard algorithm of CS. So the population of improved CS algorithm (Algorithm 2) can be structured in terms of three

Types of cuckoos:

1. A cuckoo, seeking (from the best position) areas which may contain new solutions that are much better than the solution of an individual can be randomly selected in the population;

2. A fraction p_a of cuckoos seek new solutions far from the best solution;

3. A fraction p_c of cuckoos search for solutions from the current position and try to improve them. They move from one region to another via Levy flights to locate the best solution in each region without being trapped in a local optimum.

The goal of this improvement is to strengthen intensive search around the best solutions of the population, and at the same time, randomization should be properly used to explore new areas using Levy flights. Thus, an extension to the standard CS is the addition of a method that handles the fraction ‘p_c’ of smart cuckoos.

We can expect that the new category of the cuckoo makes it possible for CS to perform more efficiently with fewer iterations. It gives better resistance against any potential traps and stagnation in local optima in the case of TSP. This allows to the adaptation of CS to TSP more control over the intensification and diversification with fewer parameters.

Algorithm 2 Improved Cuckoo Search

```
1: Objective function $f(x), x = (x_1, \dots, x_d)^T$
2: Generate initial population of $n$ host nests $x_i$ $(i = 1, \dots, n)$
3: while ($t$ <MaxGeneration) or (stop criterion) do
4:     Start searching with a fraction ($p_c$) of smart cuckoos
5:     Get a cuckoo randomly by Lévy flights
6:     Evaluate its quality/fitness $F_i$
7:     Choose a nest among $n$ (say, $j$) randomly
8:     if ($F_i > F_j$) then
9:         replace $j$ by the new solution;
10:    end if
11:    A fraction ($p_a$) of worse nests are abandoned and new ones are built;
12:    Keep the best solutions (or nests with quality solutions);
13:    Rank the solutions and find the current best
14: end while
15: Postprocess results and visualization
```

One of the objectives in extending CS to solve the traveling salesman problem (TSP) is to keep its main advantages and integrate these advantages into the discrete version of improved CS. The process of adapting CS to TSP focuses mainly on the reinterpretation of terminology used in the basic CS. CS and its inspiration sources can be structured and explained in the following five main elements: egg, nest, objective function, search space, Levy flights.

4.1 The egg

If we assume that a cuckoo lays a single egg in one nest, we can give eggs the following properties:

- An egg in a nest is a solution represented by one individual in the population;
- An egg of the cuckoos is a new solution candidate for a place/location in the population.

We can say that an egg is the equivalent of a hamiltonian cycle. Here, we neglect the need to take a departure city for all circuits and also the direction of the tour taken by the salesman.

4.2 The nest

In CS, the following features can be imposed concerning a nest:

- The number of nests is fixed;
- A nest is an individual of the population and the number of nests is equal to the size of the population;
- An abandoned nest involves the replacement of an individual of the population by a new one. By the projection of these features on TSP, we can say that a nest is shown as an individual in the population with its own hamiltonian cycle. Obviously, a nest can have multiple eggs for future extensions. In the present paper, each nest contains only one egg, for simplicity.

4.3 Objective function Each solution in the search space is associated with a numeric objective value. So the quality of a solution is proportional to the value of the objective function. In CS, a nest egg of better quality will lead to new generations. This means that the quality of a cuckoo's egg is directly related to its ability to give a new cuckoo. For the traveling salesman problem, the quality of a solution is related to the length of the hamiltonian cycle. The best solution is the one with the shortest hamiltonian cycle.

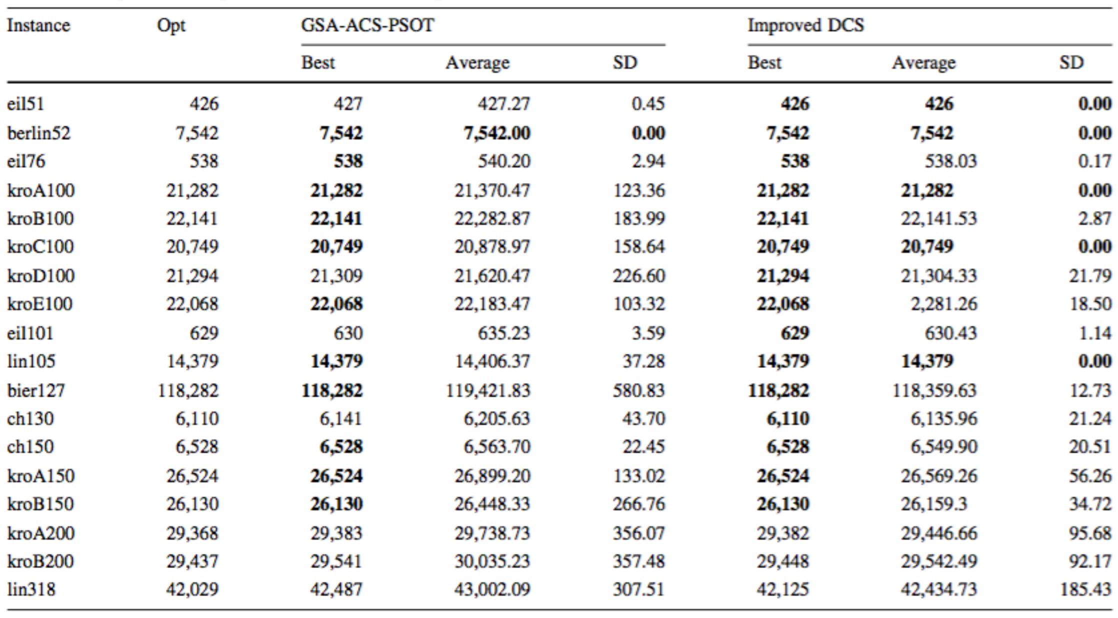
4.4 Search space In the case of two dimensions, the search space represents the positions of potential nests. These positions are ðx; yÞ 2 R R. To change the position of a nest, we only have to modify the actual values of its coordinates. It is obvious that moving nests or locations of the nests does not impose real constraints. This is the case in most continuous optimisation problems, which can be considered as an advantage that avoids many technical obstacles such as the representation of the coordinates in the solution space of TSP, especially in the mechanism for moving a solution from one neighbourhood to another. The coordinates of cities are fixed coordinates of the visited cities; however, the visiting order between the cities can be changed.

4.5 Levy Flights

Levy flights have as a characteristic of an intensive search around a solution, followed by occasional big steps in the long run. According to Yang and Deb , in some optimisation problems, the search for a new best solution is more efficient via Levy flights. In order to improve the quality of search, we will associate the step length to the value generated by Levy flights as outlined in the standard CS.

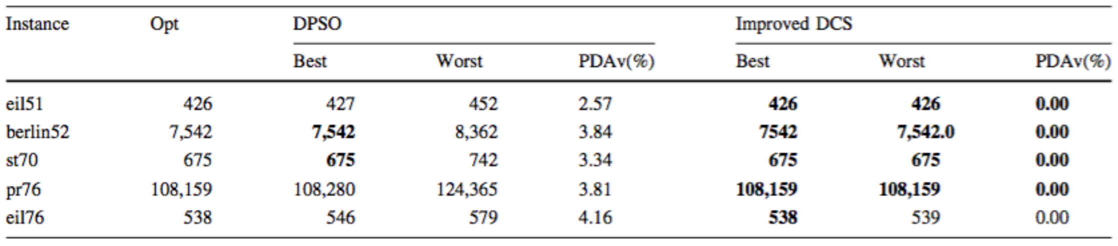
Comparison Discrete cuckoo algorithm with few other algorithms

Table 4 Comparison of experimental results of the improved DCS with **GSA-ACS-PSOT** [6]



Instance	Opt	GSA-ACS-PSOT			Improved DCS		
		Best	Average	SD	Best	Average	SD
eil51	426	427	427.27	0.45	**426**	**426**	**0.00**
berlin52	7,542	**7,542**	**7,542.00**	**0.00**	**7,542**	**7,542**	**0.00**
eil76	538	**538**	540.20	2.94	**538**	538.03	0.17
kroA100	21,282	**21,282**	21,370.47	123.36	**21,282**	**21,282**	**0.00**
kroB100	22,141	**22,141**	22,282.87	183.99	**22,141**	22,141.53	2.87
kroC100	20,749	**20,749**	20,878.97	158.64	**20,749**	**20,749**	**0.00**
kroD100	21,294	21,309	21,620.47	226.60	**21,294**	21,304.33	21.79
kroE100	22,068	**22,068**	22,183.47	103.32	**22,068**	2,281.26	18.50
eil101	629	630	635.23	3.59	**629**	630.43	1.14
lin105	14,379	**14,379**	14,406.37	37.28	**14,379**	**14,379**	**0.00**
bier127	118,282	**118,282**	119,421.83	580.83	**118,282**	118,359.63	12.73
ch130	6,110	6,141	6,205.63	43.70	**6,110**	6,135.96	21.24
ch150	6,528	**6,528**	6,563.70	22.45	**6,528**	6,549.90	20.51
kroA150	26,524	**26,524**	26,899.20	133.02	**26,524**	26,569.26	56.26
kroB150	26,130	**26,130**	26,448.33	266.76	**26,130**	26,159.3	34.72
kroA200	29,368	29,383	29,738.73	356.07	29,382	29,446.66	95.68
kroB200	29,437	29,541	30,035.23	357.48	29,448	29,542.49	92.17
lin318	42,029	42,487	43,002.09	307.51	42,125	42,434.73	185.43

Table 5 Comparison of experimental results of the improved DCS with **DPSO** [27]



Instance	Opt	DPSO			Improved DCS		
		Best	Worst	PDAv(%)	Best	Worst	PDAv(%)
eil51	426	427	452	2.57	**426**	**426**	**0.00**
berlin52	7,542	**7,542**	8,362	3.84	**7542**	**7,542.0**	**0.00**
st70	675	**675**	742	3.34	**675**	**675**	**0.00**
pr76	108,159	108,280	124,365	3.81	**108,159**	**108,159**	**0.00**
eil76	538	546	579	4.16	**538**	539	0.00

The column 'opt' shows the optimal solution length taken from the TSPLIB, the column 'best' shows the length of the best solution found by each algorithm, the column 'average' gives the average solution length of the 30 independent runs of each algorithm, the column 'worst' shows the length of the worst solution found by each algorithm, the column 'PDav(%)' denotes the percentage deviation of the average solution length over the optimal solution length of 30 runs, the column

‘PDbest(%)’ gives the percentage deviation of the best solution length over the optimal solution length of 30 runs, and the column ’time’ shows the average time in seconds for the 30 runs. The percentage deviation of a solution to the best known solution (or optimal solution if known) is given by the formula:

$$PDsolution\ (\%) = \frac{solutionlength - bestknown\ solution\ length}{best\ known\ solution\ length} \times 100$$

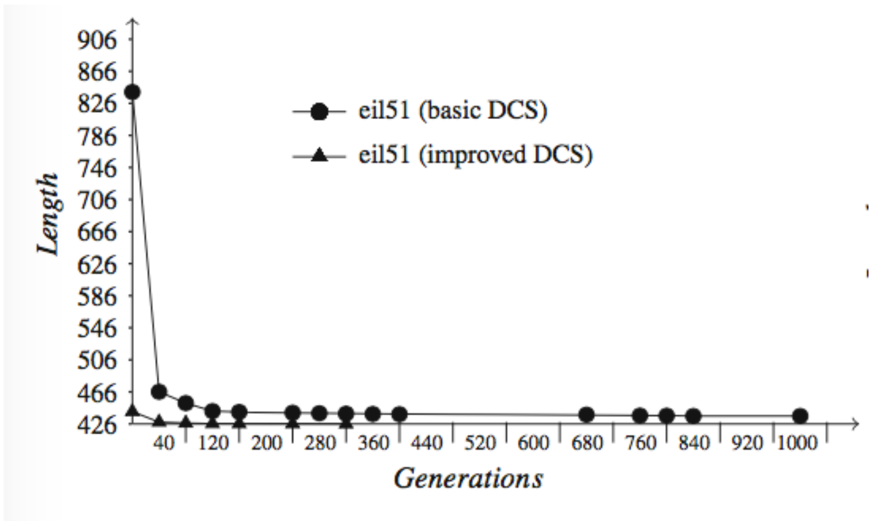


Fig. 3 Average length of the best solutions of 10 runs for eil51(opt = 426) and lin318(opt = 42029)

References

1. www.math.uwaterloo.ca/**tsp/history/** accessed 11th june 2015

2. LAPORTE, G. 2006. A short history of the traveling salesman problem. *Canada Research Chair in Distribution Management, Centre for Research on Transportation (CRT) and GERAD HEC Montre ́al, Canada*.
3. D. J. Rosenkrantz, R. E. Stearns, and P. M. Lewis, "An analysis of several heuristics for the traveling salesman problem," *SIAM J. Comput.*, vol. 6, pp. 563–581, 1977
4. Ant Colony System: A Cooperative Learning Approach to the Traveling Salesman Problem by Marco Dorigo, *Senior Member, IEEE*, and Luca Maria Gambardella, *Member, IEEE*
5. M. Dorigo, "Optimization, learning and natural algorithms," Ph.D. dissertation, DEI, Politecnico di Milano, Italy, 1992 (in Italian).
6. Proceedings of the First International Conference On Genetic Algorithms and Their Applications ,July 24-26 ,1985 at the Carnegie Mellon University Pittsburgh ,PA
7. Discrete cuckoo search algorithm for the travelling salesman problem 2014 aziz ouaarab belaid ahmiod xin she yang
8. Genetic algorithms for the travelling salesman problem 1996 Jean Yves Potvin
9. Solving travelling salesman problem based on genetic simulated annealing ant colony system wth particle swam optimization techniques 2011 shyi-ming chen , chih –yao chien
10. Simmulated annealing Dimitris Bertsimas and john Tsitklis 1993
11. Cuckoo search via levy flights 2009 xin she yang ,suas Deb
12. Genetic algorithms and travelling salesman problem 1996 Sangit chatterjee Cecilia Carrera Lucy A Lynch