Università degli studi di Milano-Bicocca

Decision Models

Final Project

An Hybrid Metaheuristic Approach to the Traveling Salesman Problem

Authors:
Dario Bertazioli-847761-d.bertazioli@campus.unimib.it
Fabrizio D'Intinosante-838866-f.dintinosante@campus.unimib.it
Massimiliano Perletti-847548-m.perletti2@campus.unimib.it

June 18, 2019

Abstract

The ABSTRACT is not a part of the body of the report itself. Rather, the abstract is a brief summary of the report contents that is often separately circulated so potential readers can decide whether to read the report. The abstract should very concisely summarize the whole report: why it was written, what was discovered or developed, and what is claimed to be the significance of the effort. The abstract does not include figures or tables, and only the most significant numerical values or results should be given. The ABSTRACT is not a part of the body of the report itself. Rather, the abstract is a brief summary of the report contents that is often separately circulated so potential readers can decide whether to read the report. The abstract should very concisely summarize the whole report: why it was written, what was discovered or developed, and what is claimed to be the significance of the effort. The abstract does not include figures or tables, and only the most significant numerical values or results should be given.

Contents

1 Introduction

The problem: the travelling salesman problem (TSP) is an algorithmic problem tasked with finding the shortest route between a set of points and locations that must be visited. In the problem statement, the points are the cities a salesperson might visit. The salesman‘s goal is to keep the distance travelled as low as possible. TSP has been studied for decades and several solutions have been theorized. The simplest solution is to try all possibilities, but this is also the most time consuming and expensive method. Many solutions use heuristics, which provides probability outcomes. It must be considered that the results are approximate and not always optimal.

Our approach: in this project we tried to apply two meta-heuristics named *Ant Colony Optimization* and *Genetic Algorithm*, implementing their "classical" version and a custom one integrating *Reinforcement Learning Algorithm*, namely *Q-learning* and *Clustering algorithm*, in particular *K-Means* respectively for the first and the second one.

2 Theoretical context

In this work we focus on two different approaches to the TSP: implementing some algorithms of the "Ant-type", and some Evolutionary Algorithms.

2.1 Ant Family Algorithms

The former approach is based on the exploitation of a set of algorithms called "Ant-Family".

2.1.1 Ant Colony Optimization

The first type of "Ant-like" algorithm we implement is the **Ant Colony Optimization** (ACO) algorithm.

The procedure draws inspiration from a "real" Ant Colony. In nature, such a system is known to accomplish some difficult tasks, being beyond the capabilities of a single ant, exploiting the individuals collaborating with each other.

In particular, ACO algorithm is based on foraging behaviour of some ant species. This behaviour can be summed up as their ability to find the shortest paths between a source of food and their nest. The cooperation among the ants has inspired researchers to apply a similar collaboration based algorithm to those problems whose solutions can be formulated as a least cost path between an origin and a destination. Since most optimisation problems might have such a formulation, those kind of algorithms are pretty interesting.

The first ACO algorithm, Ant System, was proposed by Dorigo [5, 6, 7, 8, 9].

It consists in a multi-agent approximate approach that it is said it can produce good-quality solutions in a reasonable time for combinatorial optimisation problems [5]. The author demonstrate the performance of this algorithm on Travelling Salesman Problem (TSP) [6].

Regarding the basic mechanism of ACO, here follows a quick biological explaination. Ant species are almost blind, thus they interact with the environment and communicate with each other expliting the hormones they release. In partiuclar some ant species use a special kind of hormone called **pheromone**: they lay pheromone trails on the paths they explore, these traces act as stimuli and other ants belonging to the colony are attracted to follow the paths that have relatively more tracked. Due to this mechanism, an individual who is following a path because of the pheromone trail also reinforces it by dropping its own pheromone too.

Thus, the more ants follow a specific path, the more likely that path becomes to be followed by the ants in the colony [5, 8, 9].

ACO algorithm makes use of ant-like agents called artificial ants, that construct their solutions collaboratively by sharing their experience on the quality of solutions that were generated so far.

The pheromone trails play a leading role in the utilization of collective experience. The solutions are built iteratively. Artificial ants have "memory" to store the path they followed while constructing their solutions. Exploiting such a memory, typically (even though depending on the speficic class of ant colony algorithm) artificial ants do not deposit the pheromone until they have constructed their solution. Then, They determine the amount of pheromone according to the quality of their solution and upload the pheromone matrix (the data structure in which the pheromone amount for each part of the total path is stored). Automatically, the paths belonging to better solutions, receive more pheromone.

In iteratively building a solution (a total path) for a single ant, a local stochastic transition policy is typically applied, stating how to decide the next node to visit in a graph. Artificial ants make their decisions and transitions to their next state in discrete time steps, deciding whether to follow the main trails, or to random explore a new path (in our implementation, such a decision is made by a random number generation and imposing a threshold). Exploration is also encouraged by a mechanism of pheromone evaporation, which prevents the colony from getting stuck into a solution corresponding to a (only) local optimum (note however that in real ant colonies pheromone evaporation is too slow to be a significant part of their search mechanism).

Summing up, Ant Colony Optimisation is a metaheuristic proposed to solve hard optimisation problems. The ACO metaheuristic, from a high-level view, is composed of 3 main stages:

- ConstructAntSolutions: the artificial ants construct their solutions. The transition policy controls the ants' next step to one of the adjacent nodes. Once the ants have completed their path, the quality of the current solution is evaluated, and used in the next step. The decision policy is based on following a probability distribution of the type [5]:

$$p_{ij}^k = \frac{[\tau_{ij}]^\alpha [\eta_{ij}]^\beta}{\sum_{l \in N_i^k} [\tau_{il}]^\alpha [\eta_{il}]^\beta} \quad \text{if } \mathrm{j} \in \mathrm{N_i^k} \tag{1}$$

  where

  - η_{ij} indicates an heuristic value specified according to the problem (in the TSP case, is equal to $1/d_{ij}$,
  - τ_{ij} is the pheromone quantity on the path between the i-th and j-th nodes,
  - α and β are the parameters used to set the relative importance of the pheromone trail and the heuristic value. As $\alpha \to 0$, the pheromone track become less important and the ants tend to choose the closest cities, resulting in a much more "greedy" search. Viceversa, when $\beta \to 0$, heuristic values are almost ignored and only the tracks are considered in the decision making.

- UpdatePheromones: the pheromone trails are adjusted based on the latest iteration of the colony search process. Two different updates happen:

  - the pheromone evaporates according to the equation:

$$\tau_{ij} = (1 - \rho)\tau_{ij} \tag{2}$$

    where ρ is the evaporation coefficient.

  - new pheromone is deposited on the followed path. The amount of pheromone to deposit is typically decided according to the quality of the particular solutions that each path belongs to:

$$\Delta\tau_{ij} = \sum_{k=1}^{m} \Delta\tau_{ij}^k \tag{3}$$

Algorithm 1 Ant Colony Optimization

Main Algorithm

```
initialize best_dist and best_path to None
for generation in generations:
    create n_ants artificial ants
    for one_ant in ants:
        make a single ant path (see Make path)
        compute the path length
        update best_dist and best_path
    update the pheromon matrix
    (local update only for child processes, according to eq. 4)
    every a certain n of iterations:
        update the global pheromone matrix
        (shared in MPI environment among master&child.)
return best_dist, best_sol
```

Make path

```
start from a vertex
add start vertex to visited nodes
for each remaining vertex:
    list the neighbors
    list the not yet visited neighb
    calculate the probability of choosing a vertex (according to eq. 1)
    choose the vertex according to probability
    add the choosen vertex to the visited list
    return the chosen vertex id
```

Local update pheromon matrix

```
for ant in ant_colony :
    for each vertex of one_ant_path :
        increase pheromon_matrix between current and next vertex of Δτ
        (according to eq. 4)
```

Global update pheromon matrix (parallelism)

```
gather from MPI env all the pheromone matrices
if process is the parent process (rank==0):
  for each element average over the n_cores matrices.
broadcast obtained pheromone matrix to the other
processes
```

Table 1: Ant Colony pseudocode

where $\Delta\tau_{ij}^k$ is the pheromone increase amount deposited by the k-th ant, which can be (e.g. in Dorigo initial work) taken either as a constant, or $\Delta\tau_{ij} = 1/L_k$, where L_k is the $k-th$ ant path lenght.
However the entity of pheromone update and it's weight on how the search will be biased towards the best solution found so far is an implementation decision.

- the following equations can be combinated in:

$$\tau_{ij} = (1-\rho)\tau_{ij} + \Delta\tau_{ij} \tag{4}$$

The pseudocode of the ACO solution is presented in Tab. 1.

2.1.2 Ant-Q Metaheuristic

In order to better understanding the working mechanism of Ant-Q Metaheuristic and to give deeper insight in our implentation (following [10]), let us introduce some theoretical hints for the context.

Hints on reinforcement learning: Reinforcement Learning (RL) is an (almost) unsupervised learning approach.

Randomly initialise Q(s, a)
Repeat for each episode
    Initialise the current state s
    While s is not terminal state
        Choose action a at the current state s according to the policy (e.g. $\varepsilon - greedy$)
        Take action a, observe reward r and next state s'
        $Q(s,a) \leftarrow Q(s,a) + \alpha\left[r + \gamma \max_{a'} Q(s',a') - Q(s,a)\right]$
        $s \leftarrow s'$;

Figure 1: The pseudo code of a typical Q-learning algorithm implementation.

It consists of an **agent** who tries to learn how to reach a goal by a continuous interaction with the environment. There is an evaluation phase where the quality of agent's actions is considered and feedbacks to the agent are given in the form a numerical reward. This type of feedback is known as evaluative feedback: in contrary of supervised learning, here the agent is not explicitly told what action is the best to take in a certain situation, whereas it should try a set of possible actions and learn the best strategy yielding the most reward itself.

In some cases, the goal state (that is, the agent reaching its objective) can be obtained only after a sequence of actions: as a result the reward is delayed (FIXME: cfr section ant q delayer reward).

Summing up, and according to [11], the RL problem can be defined as the problem of an agent interacting with a complex environment trying to maximise its long-run reward over a sequence of discrete time steps.

The agent follows a **policy** to decide on its action according to the current state and conditions.

This policy is typically a stochastic function ($\pi(s, a)$) that indicates a probability of choosing an action a given a state s. Notice that agent has the possibility to change its initial policy according to new experiences in order to achieve optimal cumulative reward over time.

The value of a state $V_\pi(s)$ is defined as the expected cumulative reward that will be obtained starting from a state s and acting according to the current policy π. In the same way, the value of a pair state-action($Q_\pi(s, a)$) is the expected return obtained starting from s with action a and then following the policy. In formulas, V is defined as:

$$V^\pi(s) = E_\pi\{\sum_i \gamma^i r_{t+1,i+1} \ {}_{|s_t=s}\} \tag{5}$$

and accordingly:

$$Q^\pi(s, a) = E_\pi\{\sum_i \gamma^i r_{t+1,i+1} \ {}_{|\ s_t=s,\ a_t=a}\} \tag{6}$$

The RL problem consists in the agent trying to find the optimal policy π^* that maximizes the value functions, obtaining thus:

$$V^*(s) = max_{a\in A(s)}(Q^{\pi^*}(s, a) \tag{7}$$

however, the policy estimation can be in general a complex problem, and an optimal policy can be obtained with various algorithms, such as Policy Iteration and Value Iteration. There are also kind of learning mechanism defined as "off-policy", because they do not exploit a proper policy procedure.

Q-Learning : is an off-policy method, meaning that it updates the values iteratively basing this process on the action that gives the maximum value (that is, such an algorithm tries to directly learn Q^* instead of learning Q_π first). In figure 1 it is shown the pseudocode of the algorithm: the agent uses a so called ϵ-greedy policy, but updating the current value estimate considering the action that provides the maximum value at the successor state instead of considering the (current-)policy-suggested action.

Ant-Q Algorithm : one of the core points of this project consists in the implementation of the Ant-Q algorithm, introduced by Gambarella [10] in collaboration with Dorigo, attempting to ameliorate the "classic" ACO performances.

In this approach, the pheromone update rule is borrowed from the Q-learning prassi:

$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha(r + \gamma[max_{a'} Q(s^{'}, a^{'}) - Q(s,a)] \ . \tag{8}$$

In particular, equation 4 is changed into the following:

$$\tau_{ij} = (1-\alpha)\tau_{ij} + \alpha(\Delta\tau_{ij} + \gamma max_{l \in N_j^k} \tau_{jl}) \ . \tag{9}$$

(FIXME: fix eq punctuation everywhere) The next action (what next node connecting to) is chosen according to:

$$s = \begin{cases} arg\ max_{a \in J_k(s)} [Q(s,a)]^{\alpha} [\eta(s,a)]^{\beta} & \text{if } q \leq q_0 \\ S & \text{otherwise} \end{cases} \tag{10}$$

where correspondingly to equation 1, η is an heuristic value associated with the inverse of distance between a pair of nodes, and α , β are weight-parameters for the interaction.

Comparing equation 9 to to the Q-learning update rule 8 , it is worth to notice that :

- equation 9 updates the pheromone value of the transition (i, j) according to the pheromone value of the next transition (j, l),
- equation 9 uses the second part of equation 8 (known as TD Error) to weight the pheromone quantity associated to the current edge with a learning rate α and a discount rate γ,
- the equation
$$\tau_{ij} = (1-\alpha)\tau_{ij} + \alpha(\gamma max_{l \in N_j^k} \tau_{jl}) \tag{11}$$
is used for the pheromon matrix update (namely a local update) during each path construction (of each ant), and it does not include the delayed reward $\Delta\tau_{ij}$,
- $\Delta\tau_{ij}$ is calculated according to the solution quality, as anticipated circa equation 3, and assigned in a "delayed" mode: thus the value of $\Delta\tau_{ij}$ for all i and j will be 0 while the ants apply the update rule 11 during their construction of the current solution. Therefore, the update rule 9 is reapplied at the completion of the current solution, but with the value of the next transition considered to be equal to zero. (thus uploading only with $\Delta\tau_{ij}$:
$$\tau_{ij} = (1-\alpha)\tau_{ij} + \alpha(\Delta\tau_{ij}) \tag{12}$$
- still according to [10] [11] $\Delta\tau_{ij}$ can be updated with an iteration best rule (update every single colony iteration) or global best (update based on the global best value.

Notice that many other attempts were made in the direction of hybridizing ACO and reinforcement learning based algorithms, some particularly interesting and successful, such as [13],[14],[12]. However, due to the few time available, we could head our force in implementing only the in-detail-described Ant-Q.

The pseudocode regarding our implementation of such an hybrid metaheuristic is reported in Tab. 2

Parallel Implementation: since both the ACO and Ant-Q are memory and computational-expensive, we suggest a parallel implementation of each algorithm.

The type of proposed algorithm is naturally easy to code in a parallel implementation, being enough to split the total number of ants into a number of sets (called *n_ants_per_core* in the code) of ants that will be distributed on each single core.

Algorithm 2 Ant-Q algorithm

```
Main Algorithm
initialize best_dist and best_path to None
for generation in generations:
    create n_ants artificial ants
    for one_ant in ants:
        make a single ant path (see Make path)
        compute the path length
        update best_dist and best_path
        update ph. matrix with delayed rewards
        (according to eq. 12)
    update the global pheromone matrix
  (shared in MPI environment among master&child.)
return best_dist, best_sol

Make path
start from a vertex
add start vertex to visited nodes
for each remaining vertex:
    list the neighbors
    list the not yet visited neighb
    generate a random number q ∈ {0, 1}
    if q ¡ q_0: (threshold)
        select next vertex according to eq. 10
    else:
        calculate the probability of choosing a vertex
        (according to eq. 1)
         choose the vertex according to probability
     add the choosen vertex to the visited list
     give local rewards (local update pheromone matrix)
     return the chosen vertex id

Local update pheromone matrix
for ant in ant_colony :
    for each vertex of one_ant_path :
        increase pheromon_matrix between current
       and next vertex of a Δτ (according to eq. 9)

Global update pheromon matrix (parallelism)
gather from MPI env all the pheromone matrices
if process is the parent process (rank==0):
  for each element average over the n_cores matrices.
broadcast obtained pheromone matrix to the other
  processes
```

Table 2: Ant-Q pseudocode

Thus, a parent process initializes the algorithm, creating n_core childs each operating on a "personal" memory area. Thus, every child has its own data structures, in particular its own pheromone matrix. The MPI/OpenMPI[1] [2] software (in partiicular mpi4py[3], a pythonic Api for OpenMPI) is used to host a shared pheromone matrix which will be updated according to the pseudocode in tab. 1,2 (as a mean-matrix), and which every "local" matrix will be update to after every iteration, so that every artificial ant in every child process will "feel" a as similar as possible pheromone effect.

[1] `https://www.open-mpi.org/`

[2] a nice introduction to MPI basic usage: `https://princetonuniversity.github.io/PUbootcamp/sessions/parallel-programming/Intro_PP_bootcamp_2018.pdf`

[3] https://mpi4py.readthedocs.io/en/stable/

2.2 Evolutionary Algorithms

2.2.1 Genetic Algorithm

The latter approach we faced along this work is the **Genetic Algorithm**, which we initially implement in its classical version. This algorithm is based on a biological metaphor: the resolution of a problem is seen as a competition among a population whose evolving individuals become better and better candidates solutions over time. A "fitness" function is used to evaluate each individual to decide whether it will contribute to the next generation. Then, in analogy with the biological metaphor (the gene transfer in sexual reproduction), a crossover operator is applied in order to generate the next generation of the population. This process, according to the evolutionary theory (Darwinism), should lead after a certain number of iterations to a much more fit ensemble of individuals representing "good" candidate solutions to the considered problem.

The pseudo code of the standard genetic algorithm is summarized in the Tab. 3, where Tm is the mutation rate that determines the rate at which the mutation operator is applied, Tp is the population size (number of chromosomes) and MaxG the number of generations used in the experiment[15]. As said evolutionary algorithms like *Genetic algorithm* uses operators inspired by natural selection such as reproduction, mutation, recombination and selection. *Genetic algorithm* is very customizable in every component; in this work we have focused our attention on the selection part. In particular we tried to apply the following selection strategies [16]:

- roulette-wheel selection;
- tournament selection.

The **roulette-wheel** selection consists in giving a weight to each chromosomes (the individuals in our population) and this weight corresponds to a portion of a roulette wheel. In this way chromosomes with higher fitness will have higher weight and a larger portion of the wheel.

$$p_i = \frac{f_i}{\sum_{j=1}^{n} f_j} \tag{13}$$

Spinning the wheel for different times allows selecting the individuals for the next generation. Finally, the roulette wheel is nothing more than a weighted sort mechanism.

The **tournament** selection, instead, consists in randomly selecting a group of individuals from the larger population and taking only the one with the highest fitness. The number of individuals competing in each tournament is commonly set to 2 but in this work we implemented a variable tournament dimension for each iteration.

The other elements, however, have been kept standard. Among these we remember:

- **crossover:** we applied the simplest crossover strategy called *single-point*;
- **mutation:** we applied the simplest mutation version called *point mutation*, with a mutation probability described by a **mutation rate**.

Finally, with the aim to explore a new variation of the standard algorithm, we try to integrate a *Clustering algorithm* named *K-Means* in order to reduce the problem dimension and improve the genetic procedure performance.

2.2.2 KGA Metaheuristic

The **K-Means Genetic Algorithm** (KGA) is composed by different phases described below.

Algorithm 3 Genetic Algorithm

```
procedure Genetic(Tm,Tp,MaxIt)
  Pop ← GeneratePopulation(Tp)
  Pop ← Evaluation(Pop)
  for i = 1 ... MaxIt do
      Pop ← Selection(Pop)
      Pop ← Crossover(Pop)
      Pop ← Selection(Pop)
      With probability Tm do:
        Pop ← Mutation(Pop)
   end for
    return the best solution in Pop
end procedure
```

Table 3: Genetic Algorithm pseudocode

Algorithm 4 K-Means Algorithm

```
Set the K cluster centers randomly;
repeat
  for each vertex do
      Calculate distance measure to each cluster;
      Assign it to the closest cluster;
  end
  recompute the cluster centers positions;
until stop criteria are met;
```

Table 4: K-Means pseudocode

Clustering with K-Means: At first, specifically for the TSP problem, we need to cluster our cities into close groups with a clustering method.

The *K-Means* method is designed to partition a set of data into K classes with K chosen as desired. This method constructs partitions of the data-matrix so that the squared Euclidean distance between the row vector for any object and the centroid vector of its respective cluster is at least as small as the distances to the centroids of the remaining clusters. The centroid of cluster C_k is a point in P-dimensional space found by averaging the values on each variable over the objects within the cluster. For instance, the centroid value for jth variable in cluster C_k is

$$\bar{x}_j^{(k)} = \frac{1}{n_k} \sum_{i \in C_k} x_{ij} \tag{14}$$

and the complete centroid vector for cluster C_k is given by[17]

$$\bar{x}^{(k)} = (\bar{x}_1^{(k)}, \bar{x}_2^{(k)}, \ldots, \bar{x}_p^{(k)}) \tag{15}$$

Finally, K-Means clustering algorithm is presented in Tab: 4.

Intra-group evolution operation: The aim of the intra-group evolution operation is to find the shortest path for the given vertices in each cluster. GA is performed in each cluster aiming to obtain an approximate solution by a couple of genetic operations like selection, crossover, and mutation. Running the GA algorithm on smaller portion of original data allows to improve performance and reach better solutions. Eventually all those clusters could be handled parallel. The result of this step is tours $T_1, T_2, \ldots, T_k$ for clusters $C_1, C_2, \ldots, C_k$.

Algorithm 5 KGA

input an TSP;
K-Means is adopted to cluster the TSP into k *sub-problems*
For each sub-prob $i = 1$ to k, do:
  repeat
    GA procedure
  until *stop criteria are met*
  Output shortest path for sub-problem i;
End
Seek for the best combining seq S with GA
Combine all those shortest path into one tour
Output the shortest whole travelig tour.

Table 5: KGA pseudocode

Inter-group connection: In the last step, what we have obtained is the shortest path between the given vertices in each cluster. With the aim of reconstruct the whole shortest path we need to connect properly every cluster to the others. Connect two clusters determine which edges will be deleted from the adjacent shortest path among each cluster, and which edges will be linked for combining two adjacent clusters into one. Assuming i and j are two closest vertices between two clusters G_i and G_j for $G_i, i-1$ and $i+1$ are two adjacent vertices of i, and the same to $G_j, j-1$ and $j+1$ are two adjacent vertices of j. Given G_i and G_j, in order to combine the two clusters into one, we need to select two vertices $i \in i'$ and $j \in j'$ for deleting and linking edges. With the aim to apply this strategy we refer to Eq. 16

$$\{i^*, j^*\} = \text{argmin}_{i',j'} \begin{cases} d_{ij} + d_{i'j'} - d_{ii'} - d_{jj'} \\ d_{ij'} + d_{i'j} - d_{ii'} - d_{jj'} \end{cases} \tag{16}$$

where $i' \in \{i-1; i+1\}, j' \in \{j-1; j+1\}$. Following this strategy, the first two clusters are combined into one, then the new generated cluster combines with the third cluster, and so on, step by step. At last, all those clusters are joined into one tour, and the shortest whole traveling tour is derived. The whole process of the KGA is listed in Tab. 5 [18],[19].

3 Datasets

The datasets used in this work are taken from `https://wwwproxy.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/tsp/`, a large source of TSP datasets largely cited in literature. There are datasets of variable dimension and for everyone is also available the optimal solution so that is possible for us to compare our results with the optimal one. Every solution is available at `https://wwwproxy.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/STSP.html`. Every dataset is composed by a list of "*cities*" with two coordinates points; the only preprocessing we applied was to compute a matrix containing the distance between every point and the other ones.

4 The Methodological Approach

4.1 Ant Algorithms

Regarding the Ant-Family blabla

4.2 Evolutionary Algorithms

Our implementation of the genetic blabla

5 Results and Evaluation

The Results section is dedicated to presenting the actual results (i.e. measured and calculated quantities), not to discussing their meaning or interpretation. The results should be summarized using appropriate Tables and Figures (graphs or schematics). Every Figure and Table should have a legend that describes concisely what is contained or shown. Figure legends go below the figure, table legends above the table. Throughout the report, but especially in this section, pay attention to reporting numbers with an appropriate number of significant figures.

6 Discussion

The discussion section aims at interpreting the results in light of the project's objectives. The most important goal of this section is to interpret the results so that the reader is informed of the insight or answers that the results provide. This section should also present an evaluation of the particular approach taken by the group. For example: Based on the results, how could the experimental procedure be improved? What additional, future work may be warranted? What recommendations can be drawn?

7 Conclusions

Conclusions should summarize the central points made in the Discussion section, reinforcing for the reader the value and implications of the work. If the results were not definitive, specific future work that may be needed can be (briefly) described. The conclusions should never contain "surprises". Therefore, any conclusions should be based on observations and data already discussed. It is considered extremely bad form to introduce new data in the conclusions.

References

[1] Blum, C., and Dorigo, M. The Hypercube framework for Ant Colony Optimization. In:IEEE Transactions on Systems, Man, and Cybernetics, Vol. 34, No. 2, April 2004.

[2] Buşoniu, L., Schutter, B., and Babuska R. Learning and Coordination in Dynamic Multiagent Systems. Technical Report submitted to Delft University of Technology, Delft Center for Systems and Control, Netherlands, October 2005.

[3] Cordon, O., Herrera, F., Fernandez de Viana, I., and Moreno, L. A New ACO Model Integrating Evolutionary Computation Concepts: The Best-Worst Ant System. In: Proc. of ANTS'2000. From Ant Colonies to Artificial Ants: Second International Workshop on Ant Algorithms, Brussels, Belgium, September 7-9, pp. 22-29, 2000.

[4] Cordon, O., Herrea, F., and Stützle, T. A Review on The Ant Colony Optimization Metaheuristic: Basis, Models and New Trends. In: Mathware and Soft Computing. Vol. 9, pp.141-175, 2002.

[5] Dorigo, M., Maniezzo, V., and Colorni, A. Ant System: Optimization by a Colony of Cooperating Agents. In: IEEE Transactions on Systems, Man, and Cybernetics. Vol. 26, No. 1, 1996.

[6] Dorigo, M., and Gambardella, L. M. Ant Colony System: A Cooperative Learning Approach to the Travelling Salesman Problem. In: IEEE Transactions on Evolutionary Computation, Vol.1 No.1, pp.53-66, April 1997.

[7] Dorigo, M., Di Caro, G., and Gambardella, L. M. Ant algorithms for discrete optimisation. In: Artificial Life 5(2),pp.137-172, April 1999.

[8] Dorigo, M., and Stützle, T. Ant Colony Optimization, MIT Press, Cambridge, MA, 2004.

[9] Dorigo, M., Birattari, M., and Stützle, T. Ant Colony Optimization: Artificial Ants as a Computational Intelligence Technique. In: IEEE Computational Intelligence Magazine, November 2006.

[10] Gambardella, L. M. and Dorigo, M. Ant-Q: A Reinforcement Learning Approach to the Traveling Salesman Problem. In: Proc. ML-95, 12th Int. Conf. Machine Learning. Palo Alto, CA: Morgan Kaufmann, pp. 252–260, 1995.

[11] Sutton, R. S., and Barto, A. G. Reinforcement Learning: An Introduction. MIT Press, Cambridge, MA, 1998.

[12] Sun, R., Tatsimu, S., Zhao, G., Multiagent Reinforcement Learning with an Improved Ant Colony System. In:IEEE Transactions on Systems, Man, and Cybernetics, Vol.3, pp.1612-1617, 2001.

[13] Miagkikh, V. and Punch, W. F. An Approach to Solving Combinatorial Optimization Problems Using a Population of Reinforcement Learning Agents. In: Genetic and Evolutionary Computation Conference, pp. 1358-1365, 1999.

[14] Monekosso, N., and Remagnino, P., The Analysis and Performance Evaluation of the Pheromone-Q-learning Algorithm. In: Expert Systems, Vol.21, No.2, pp.80-91, May 2004.

[15] Chagas De Lima Junior, Francisco & Neto, Adriao Duarte & Melo, J.D.. (2010). Hybrid Metaheuristics Using Reinforcement Learning Applied to Salesman Traveling Problem. 10.5772/13343.

[16] Razali, Noraini Mohd and John Geraghty. "Genetic Algorithm Performance with Different Selection Strategies in Solving TSP." (2011).

[17] Steinley, D. (2006), K-means clustering: A half-century synthesis. British Journal of Mathematical and Statistical Psychology, 59: 1-34.

[18] L. Tan, Y. Tan, G. Yun and Y. Wu, ”Genetic algorithms based on clustering for traveling salesman problems,” 2016 12th International Conference on Natural Computation, Fuzzy Systems and Knowledge Discovery (ICNC-FSKD), Changsha, 2016, pp. 103-108.

[19] Krishna, K and Murty, Narasimha M (1999) Genetic K-Means Algorithm. In: IEEE Transactions on Systems Man And Cybernetics-Part B: Cybernetics, 29 (3). pp. 433-439.