13

Hybrid Metaheuristics Using Reinforcement Learning Applied to Salesman Traveling Problem

Francisco C. de Lima Junior[1], Adrião D. Doria Neto[2] and Jorge Dantas de Melo[3]
University of State of Rio Grande do Norte,
Federal University of Rio Grande do Norte Natal – RN,
Brazil

1. Introduction

Techniques of optimization known as metaheuristics have achieved success in the resolution of many problems classified as NP-Hard. These methods use non deterministic approaches that reach very good solutions which, however, don't guarantee the determination of the global optimum. Beyond the inherent difficulties related to the complexity that characterizes the optimization problems, the metaheuristics still face the dilemma of exploration/exploitation, which consists of choosing between a greedy search and a wider exploration of the solution space. A way to guide such algorithms during the searching of better solutions is supplying them with more knowledge of the problem through the use of a intelligent agent, able to recognize promising regions and also identify when they should diversify the direction of the search. This way, this work proposes the use of Reinforcement Learning technique – Q-learning Algorithm (Sutton & Barto, 1998) - as exploration/exploitation strategy for the metaheuristics GRASP (Greedy Randomized Adaptive Search Procedure) (Feo & Resende, 1995) and Genetic Algorithm (R. Haupt & S. E. Haupt, 2004). The GRASP metaheuristic uses Q-learning instead of the traditional greedy-random algorithm in the construction phase. This replacement has the purpose of improving the quality of the initial solutions that are used in the local search phase of the GRASP, and also provides for the metaheuristic an adaptive memory mechanism that allows the reuse of good previous decisions and also avoids the repetition of bad decisions. In the Genetic Algorithm, the Q-learning algorithm was used to generate an initial population of high fitness, and after a determined number of generations, where the rate of diversity of the population is less than a certain limit L, it also was applied to supply one of the parents to be used in the genetic crossover operator. Another significant change in the hybrid genetic algorithm is an interactive/cooperative process, where the Q-learning algorithm receives an additional update in the matrix of Q-values based on the current best solution of the Genetic Algorithm. The computational experiments presented in this work compares the results obtained with the implementation of traditional versions of GRASP metaheuristic and Genetic Algorithm, with those obtained using the proposed hybrid

methods. Both algorithms had been applied successfully to the symmetrical Traveling Salesman Problem, which was modeled as a Markov decision process.

2. Theoretical foundation

2.1 GRASP metaheuristic

The Metaheuristic Greedy Randomized Adaptive Search Procedure - GRASP (Feo & Resende, 1995), is a multi-start iterative process, where each iteration consists of two phases: constructive and local search. The constructive phase builds a feasible solution, whose neighbourhood is investigated until a local minimum is found during the local search phase. The best overall solution is kept as the final solution.

Each iteration of the construction phase let the set of candidate elements be formed by all elements that can be incorporated in to the partial solution under construction without destroying its feasibility. The selection of the next element for incorporation is determined by the evaluation of all candidate elements according to a greedy evaluation function $g(c)$, where c is a candidate element to compose the solution.

The evaluation of the elements by function $g(c)$ leads to the creation of a restricted candidate list - *RCL* formed by the best elements, i.e., those whose incorporation to the current partial solution results in the smallest incremental cost (this is the greedy aspect of the algorithm in the minimization case). Once the selected element is incorporated into the partial solution, the candidate list is updated and the incremental costs are revaluated (this is the adaptive aspect of the heuristic).

The probabilistic aspect of GRASP is given by the random choice of one of the elements of the *RCL*, not necessarily the best, except in the case of the *RCL*, which has unitary size. In this case, the selection criterion is reduced to the greedy criterion. The improvement phase typically consists of a local search procedure aimed at improving the solution obtained in the constructive phase, since this solution may not represent a global optimum.

In GRASP metaheuristics, it is always important to use a local search to improve the solutions obtained in the constructive phase. The local search phase works in an iterative way, replacing successively the current solution by a better one from its neighbourhood. Thus, the success of the local search algorithm depends on the quality of the neighbourhood chosen (Feo & Resende, 1995). This dependence can be considered a disadvantage of the GRASP metaheuristic.

The GRASP metaheuristic presents advantages and disadvantages:

Advantages:

- Simple implementation: greedy algorithm and local search;
- Some parameters require adjustments: restrictions on the candidate list and the number of iterations.

Disadvantages:

- It depends on good initial solutions: since it is based only on randomization between iterations, each iteration benefits from the quality of the initial solution;
- It does not use memory of the information collected during the search.

This work explores the disadvantages of the GRASP metaheuristic replacing the random-greedy algorithm of the constructive phase by constructive algorithms that use the Q-learning algorithm as an exploitation/exploration strategy aimed at building better initial solutions. The pseudo code of the traditional GRASP algorithm is presented in the Fig. 1,

where D corresponds to the distance matrix for the *TSP* instances, α_g[1] is the control parameter of the restricted candidate list - *RCL*, and Nm the maximal number of interactions.

Algorithm 1 Traditional GRASP Algorithm

```
1: procedure GRASP(D, α_g, Nm)
2:     f(S*) ← +∞
3:     while i ≤ Nm do
4:         S ← RandomGreedy(D, α_g)
5:         S' ← LocalSearch(S, Near(S))
6:         if f(S') < f(S*) then
7:             S* = S'
8:         end if
9:         i ← i + 1
10:    end while
11:    return S*
12: end procedure
```

Fig. 1. Traditional GRASP Algorithm

In the GRASP metaheuristic, the restricted candidate list, specifically the α_g parameter, is practically the only parameter to be adjusted. The effect of the choice of α_g value, in terms of quality solution and diversity during the construction phase of the GRASP, is discussed by Feo and Resende (Feo & Resende, 1995). Prais and Ribeiro (Prais & Ribeiro, 1998) proposed a new procedure called reactive GRASP, for which the α_g parameter of the restricted candidate list is self-adjusted according to the quality of the solution previously found.
In the reactive GRASP algorithm, the α_g value is randomly selected from a discreet set containing m predetermined feasible values:

$$\Psi = \left\{\alpha_g^1,...,\alpha_g^m\right\} \tag{1}$$

The use of different α_g values at different iterations allows the building of different Restrict Candidate Lists - *RCL*, possibly allowing the generation of distinct solutions that would not be built through the use of a single fixed α_g value.
The choice of α_g in the set Ψ is made using a probability distribution p_i, $i=1,...,m$, which is periodically updated by the so-called absolute qualification rule (Prais & Ribeiro, 1998), which is based on the average value of the solutions obtained with each α_g value and is computed as follows:

$$q_i = \frac{(F(S^*))^\delta}{A_i} \tag{2}$$

for all $i=1,...,m$, where $F(S^*)$ is the value of the overall best solution already found and δ is used to explore the updated values of probability p_i. Using q_i, the probability distribution is given by:

1 The index g is used here to differ from the αg parameter used in the Q-learning algorithm in this paper.

$$p_i = q_i \Big/ \sum_{j=1}^{m} q_j \tag{3}$$

In this paper, the reactive GRASP is presented as a more robust version of the traditional GRASP algorithm, and its performance will be compared with the new method proposed.

2.2 Genetic algorithm

Genetic algorithms (GA) are based on a biological metaphor: they see the resolution of a problem as a competition among a population of evolving candidate problem solutions. A "fitness" function evaluates each solution to decide whether it will contribute to the next generation of solutions. Then, through analogous operations to gene transfer in sexual reproduction, the algorithm creates a new population of candidate solutions. At the beginning of a run of a genetic algorithm, a large population of random chromosomes is created. Each one, when decoded, will represent a different solution to the problem at hand. Some of the advantages of a GA (R. Haupt & S. E. Haupt, 2004) include the following:

- It optimizes with continuous or discrete variables;
- Does not require derivative information;
- Simultaneously searches through wide sampling of the cost surface;
- Deals with a large number of variables and is well suited for parallel computers;
- Optimizes variables with extremely complex cost surfaces (they can jump out of a local minimum);
- Provides a list of optimum variables, not just a single solution;
- May encode the variables so that the optimization is done with the encoded variables; and
- Works with numerically generated data, experimental data, or analytical functions.

A typical genetic algorithm presents the following operators:

- Crossover - this operator is used to vary the population of the chromosomes from one generation to the next. It selects the two fittest chromosomes in the population and produces a number of offspring. There are several crossover techniques; in this work, we will use the one-point crossover.
- Selection - this operator replicates the most successful solutions found in a population at a rate proportional to their relative quality, which is determined by the fitness function. In this paper we will use the roulette wheel selection technique for this operator.
- Mutation - used to maintain genetic diversity from one generation of a population of chromosomes to the next. It selects a position of an offspring chromosome with small probability and changes the value based on the problem model; for example, in this work, the mutation operator modifies the offspring chromosome simply by changing the position of a gene.

The pseudo code of the standard genetic algorithm is summarized in the Fig. 2, where *Tc* is the crossover rate or parameter that determines the rate at which the crossover operator is applied, *Tm* is the equivalent for the mutation rate, *Tp* is the population size (number of chromosomes) and *MaxG* the number of generations used in the experiment.

Algorithm 2 Genetic Algorithm

```
1: procedure GENETIC(Tc, Tm, Tp, MaxG)
2:     Pop ← GeneratePopulation(Tp)
3:     Pop ← Evaluation(Pop)
4:     for i = 1...MaxG do
5:         Pop ← Selection(Pop)
6:         With probability Tc do:
           Pop ← Crossover(Pop)
7:         With probability Tm do:
           Pop ← Mutation(Pop)
8:     end for
9:     return the best solution in Pop
10: end procedure
```

Fig. 2. Traditional Genetic Algorithm

2.3 Reinforcement learning

The reinforcement learning is characterized by the existence of an agent which should learn the behaviours through trial-and-error interactions in a dynamic environment. In the interaction process the learner agent, at each discrete time step t receives from the environment some denominated representation state. Starting from state $s_t \in S$ (S is the set of possible states), the learner agent chooses an action $a_t \in A$ (A is the set of actions available in state s_t). When the learner agent chooses an action, receives a numerical reward r_{t+1}, which represent the quality of the action selected and the environment response to that action presenting s_{t+1}, a new state of the environment.

The agent's goal is to maximize the total reward it receives along the process, so the agent has to exploit not only what it already knows in order to obtain the reward, but also explore the environment in order to select better action in the future (Sutton & Barto, 1998).

There are some classes of methods for solving the reinforcement learning problem, such as: Dynamic Programming, Monte Carlo methods and Temporal-difference learning. Each of these methods presents its characteristics. Dynamic programming methods are well developed mathematically, but require a complete and accurate model of the environment. Monte Carlo methods require no model and are very simple conceptually, but are not suited for step-by-step incremental computation. Temporal-difference methods do not require a complete model and are fully incremental, but it's more complex to mathematical analysis. Based in the characteristics and need of the problem in hand, this work will use a well-know Temporal-difference technique denominated Q-learning algorithm, which will be presented in details in the next section.

2.3.1 The Q-learning algorithm

Not all reinforcement learning algorithms need a complete modelling of the environment. Some of them do not need to have all the transition probabilities and expected return values for all the transitions of the possible environmental states. This is the case of the reinforcement learning techniques based on temporal differences (Sutton & Barto, 1998). One of these techniques is the well-known Q-learning algorithm (Watkins, 1989), considered

one of the most important breakthroughs in reinforcement learning, since its convergence for optimum Q-values does not depend on the policy applied. The updated expression of the Q value in the Q-learning algorithm is:

$$Q(s,a) = (1-\alpha_q)Q(s,a) + \alpha_q[r + \gamma \max_{a \in A} Q(s',a)] \quad (4)$$

where s is the current state, a is the action carried out in the state s, r is the immediate reward received for executing a in s, s' is the new state, γ is a discount factor ($0 \leq \gamma \geq 1$), and α_q, ($0 < \alpha_q < 1$) is the learning factor.

An important characteristic of this algorithm is that the choice of actions that will be executed during the iterative process of function Q can be made through any exploration/exploitation criterion, even in a random way. A widely used technique for such a choice is the so-called *ε-greedy* exploration, which consists of an agent to execute the action with the highest Q value with probability *1-ε*, and choose a random action with probability ε.

The Q-learning was the first reinforcement learning method to provide strong proof of convergence. Watkins showed that if each pair state-action is visited an infinite number of times and with an adjusted value, the value function Q will converge with probability *1* to Q^*. The pseudo code of the Q-learning algorithm is presented in the Fig. 3, where, r is the reward matrix, ε is the parameter of the *ε-greedy* police.

Algorithm 3 Q-learning Algorithm

```
1: procedure QLEARNING(r, α_q, ϵ, γ, NEp)
2:     Initialize Q(s, a)
3:     repeat
4:         Initialize s
5:         repeat
6:             Select a using the ϵ − greedy rule
7:             Observe the values of r and s′
8:             Q(s, a) ← Update(Q(s, a)) ▷ (Eq. 3)
9:             s = s′
10:        until number of episodes = NEp
11:    until the convergence is gotten
12:    return Q(s, a)
13: end procedure
```

Fig. 3. Q-learning Algorithm

3. Proposed hybrid methods

Metaheuristics are approximate methods that depend on good exploration/exploitation strategies based on previous knowledge of the problem and are can guide the search for an optimum solution in order avoiding local minimum. Good strategies (or good heuristics) alternate adequately between exploration and exploitation. In other words they maintain the balance between these two processes during the search for an optimum solution. The methods presented here are hybrid since they use a well-known reinforcement learning method - Q-learning algorithm - as an exploration/exploitation mechanism for GRASP and genetic algorithm metaheuristics (Lima, F. C., et al., 2007).

The Fig. 4, presents a framework of the proposed hybrid methods:

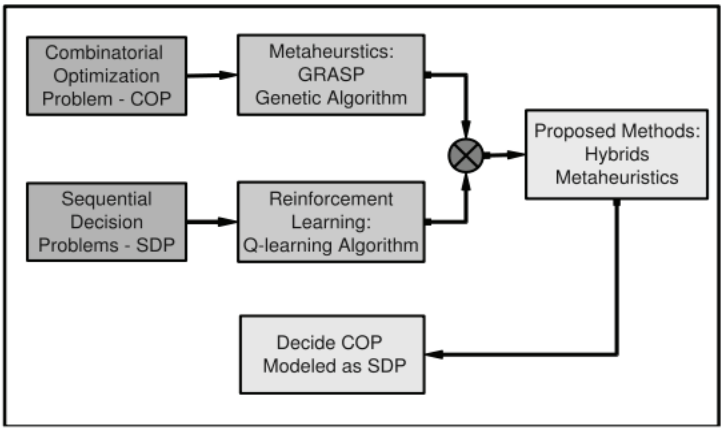


Fig. 4. Framework of the proposed hybrid methods.

The proposed methods in this section are applied to solve the symmetric traveling salesman problem - *TSP* and are modelled as a Reinforced Learning Problem.

3.1 The traveling salesman problem modeled as a sequential decision problem

The traveling salesman problem can be described as a sequential decision process, represented by the quintuplet $M= \{T, S, A, R, P\}$, in at least two different ways. For example, consider a model where the set of states S is the set of all possible solutions for *TSP* (Ramos et al, 2003). The model in this study has a high-dimensional inconvenience S, since *TSP* has a higher number of possible solutions in the symmetric case $(n-1)!/2$.

An alternative to model *TSP* as a sequential decision problem is to consider that S is formed by all the cities to solve *TSP*. In this new model, the cardinality of S is equal to the instance size of the problem. This lowers the risk of S suffering the "curse of dimensionality". In this study *TSP* is modelled as a sequential decision problem based on this second alternative.

To better understand the proposed model, consider an example of *TSP* with 5 cities shown in graph $G(V, E, W)$ of Fig. 5. V is the set of vertices, E is the set of arcs between the vertices and W is the weight associated to each arc. In graph $G(V,E,W)$, a_{12} corresponds to visiting the "city" s_2 derived from s_1, and the values associated to each arc (number between parentheses) corresponds to the distance between the "cities".

Considering graph $G(V,E,W)$, the quintuplet $M = \{T, S, A, R, P\}$, representing a sequential decision process, can be defined by *TSP* as follows:

- T: The set of decision instants is denoted by $T = \{1, 2, 3, 4, 5\}$, where the cardinality of T corresponds to the number of cities that compose a route to *TSP*.
- S: The set of states is represented by $S = \{s_1, s_2, s_3, s_4, s_5\}$, where each state s_i, $i=1,...,5$ corresponds to a city[2] at a route to *TSP*.
- A: The set of possible actions is denoted by:

[2] From this point onwards the expression "city" or "cities" indicates the association between a city of *TSP* and a state (or states) from the environment.

$$A = A(s_1) \cup A(s_2) \cup A(s_3) \cup A(s_4) \cup A(s_5)$$

$$A = \{a_{12}, a_{13}, a_{14}, a_{15}\} \cup \{a_{21}, a_{23}, a_{24}, a_{25}\} \cup \{a_{31}, a_{32}, a_{34}, a_{35}\} \cup \{a_{41}, a_{42}, a_{43}, a_{45}\} \cup \{a_{51}, a_{52}, a_{53}, a_{54}\}$$

$$A = \{a_{12}, a_{13}, a_{14}, a_{15}, a_{21}, a_{23}, a_{24}, a_{25}, a_{31}, a_{32}, a_{34}, a_{35}, a_{41}, a_{42}, a_{43}, a_{45}, a_{51}, a_{52}, a_{53}, a_{54}\} \quad (5)$$

It is important to emphasize that owing to the restriction of *TSP*, some actions may not be available when constructing a solution. To understand it more clearly, consider the following partial solution for *TSP*:

$$Sol_p : s_2 \rightarrow s_3 \rightarrow s_5 \quad (6)$$

where the decision process is in the decision instant *3* and state s_5. In this case the actions available are $A(s_5)=\{a_{51}, a_{54}\}$, since the "cities" s_2 (action choice a_{52}) and s_3 (action choice a_{53}) are not permitted to avoid repetitions in the route.

- R: S × A → ℜ: Expected Return. In TSP, elements rij are calculated using the distance between the "cities" si and sj. The return should be a reward that encourages the choice of "city" sj closest to si. Since TSP is a minimization problem, a trivial method to calculate the reward is to consider it inversely proportional to the traveling cost between the cities. That is:

$$r_{ij} = \frac{1}{d_{ij}} \quad (7)$$

where $d_{ij} > 0$ is a real number that represents the distance from "city" s_i to "city" s_j. Using the weights from the graph in Fig. 5, R is represented by:

$$R = \begin{bmatrix} 0 & 1/4 & 1/6 & 1/7 & 1/3 \\ 1/4 & 0 & 1/3 & 1/9 & 1/10 \\ 1/6 & 1/3 & 0 & 1/2 & 1/8 \\ 1/7 & 1/9 & 1/2 & 0 & 1/5 \\ 1/3 & 1/10 & 1/8 & 1/5 & 0 \end{bmatrix} \quad (8)$$

- P: S × A × S → [1, 0]: The function that defines the probability transition between states s∈S, where the elements pij(sj|si, aij) correspond to the probability of reaching "city" sj when in "city" si and choosing action aij. The values of pij are denoted by:

$$p_{ij}(s_j \mid s_i, a_{ij}) = \begin{cases} 1 & \text{if } s_j \text{ not visited} \\ 0 & \text{otherwise} \end{cases} \quad (9)$$

The quintuplet *M = {T, S, A, R, P}* defined in the graph of the Fig. 5. can be defined with no loss of generality for a generic graph *G(V, E, W)*, (with $|V|=N$) where a Hamiltonian cycle of cardinality *N* is found.

An optimum policy for *TSP*, given generic graph *G(V, E, W)*, should determine the sequence of visits to each "city" s_i, $i=1, 2,...,N$ to achieve the best possible sum of returns. In this case, the problem has a finite-time horizon, sets of states, and discrete and finite actions.

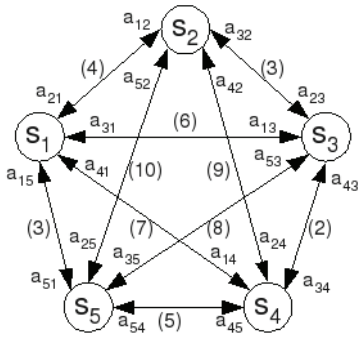


Fig. 5. Complete Graph, instance of the *TSP* with 5 cities.

3.2 The Markov property

In a sequential decision process, the environmental response in view of the action choice at any time *t+1* depends on the history of events at a time before *t+1*. This means that the dynamic of environmental behaviour will be defined by complete probability distribution (Sutton & Barto, 1998):

$$P_{ss'}^{a} = \Pr\{s_{t+1} = s', r_{t+1} = r \mid s_t, a_t, r_t, s_{t-1}, a_{t-1}, r_{t-1}, \ldots, s_1, a_1, r_1, s_0, a_0\} \tag{10}$$

where *Pr* represents the probability of s_{t+1} being *s'*, for all *s*, *a*, *r*, states, actions and past reinforcements s_t, a_t, r_t, ..., r_1, s_0, a_0.

However, if the sequential process obeys the Markov property, the environmental response at *t+1* only depends on information for state *s* and action *a* available at *t*. In other words the environment dynamic is specified by:

$$P_{ss'}^{a} = \Pr\{s_{t+1} = s', r_{t+1} = r \mid s_t, a_t\} \tag{11}$$

which qualifies it as a Markov decision process – *MDP* (Puterman, 1994).

In the traveling salesman problem, an action is chosen (deciding which city to place on the route under construction) from state s_t (the actual city in the route under construction) based on the instance between this city and the others. It is important to note that, to avoid violating the restriction imposed by *TSP* (not to repeat cities on the route), actions that would lead to states already visited should not be available in state s_t. In this context, *TSP* is a Markov decision process since all the information necessary to make the decision at *t* is available at state s_t. A list of actions not chosen during route construction could be included in state s_t to provide information on the actions available in the state.

The Markov property is fundamentally important in characterizing the traveling salesman problem as a reinforcement learning task, since the convergence of methods used in this study depends on its verification in the proposed model.

The restriction imposed on *TSP* creates an interesting characteristic in the Markov decision process associated to the problem. The fact that the sets of available actions *A(s)* for state *s* at each time instant *t* vary during the learning process, implies changes in the strategies for choosing actions in any state s_t. This means that modifications will occur in the politics used during the learning process. Fig. 6 demonstrates this characteristic.

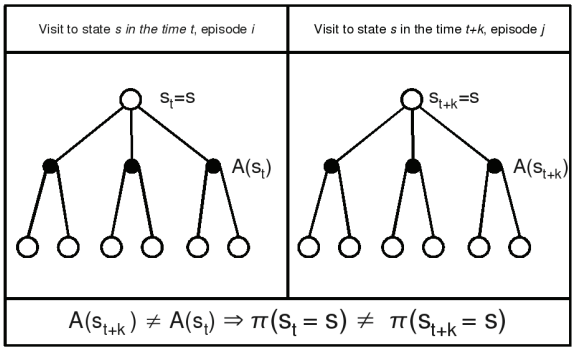


Fig. 6. Changes in the number of actions available during the decision process.

To understand more clearly, consider the following partial solution for *TSP*: The chart in Fig. 6, can be interpreted as follows: A route to *TSP* is constructed in each episode of the learning process. During construction of a route to *TSP*, at each decision instant t, a state is visited $s \in S$ and an action choice $a \in A$ is made. Since the set $A(s)$ of actions available for state s varies along time, in an episode i the set of actions available for state s_t at instant t can be different of the actions available in the same state s at time instant $t+k$ and episode j. This confirms that the Markov decision process associated to the traveling salesman problem can be modelled has a non-stationary policy.

3.3 The Q-learning algorithm implemented

Since the methods presented in this section use Q-learning, this section presents details such as: action selection policies and convergence of the algorithm when applied to the proposed problem.

3.3.1 Action selection policies for the Q-learning algorithm

When solving a reinforcement learning Problem, an action selection policy aims to determine the behaviour of the agent so that it alternates adequately between using obtained knowledge and acquiring new knowledge. This optimizes the exploration/exploitation process of the search space. More than one action selection policy is used for Q-learning to determine which policy is most appropriate to implement the proposed hybrid methods.

- The *ε-greedy* policy chooses the action with the highest expected value, compatibility defined by *(1-ε)* and random action with a probability of *ε*. Mathematically, given the matrix of Q-values $Q(s, a)$, the greedy action a^* is obtained for state s as follows:

$$a^* = \max_{a \in A(s)} Q(s,a)$$
$$\pi(s,a^*) = 1 - \varepsilon + \frac{\varepsilon}{|A(s)|} \tag{12}$$
$$\pi(s,a) = \frac{\varepsilon}{|A(s)|} \quad \forall a \in A(s) - \{a^*\}$$

where $|A(s)|$ corresponds to the number of possible actions from s, and ε is the control parameter between greed and randomness. The restriction in Eq. 12 allows Q-learning to explore the state space of the problem and is needed to find the optimum control policy.

- The *Adaptive ε-greedy* Policy is similar to the *ε-greedy* policy described. In other words, it allows the action with the highest expected value to be chosen, with probability defined by $(1 - \varepsilon)$ and a random action with probability ε. It is different and also "Adaptive", since the value of ε suffers exponential decay calculated by:

$$\varepsilon = \max\left\{v_i, v_f.b^k\right\} \tag{13}$$

where k is the episode counter of Q-learning, b is the closest value of 1 and $v_i < v_f \in [0,1]$. The algorithm initially uses high values for ε (close to v_f) and, as the value of k increases, choice ε is directed to lower values (closer to v_i). The objective is to allow more random choices to be made and to explore more the greedy aspect as the number of episodes increases.

- Based on *Visitor Count* Policy chooses the action based on a technique called Reinforcement Comparison (Sutton & Barto, 1998). In this technique actions are chosen based on the principle that actions followed by large rewards are preferred to those followed by small rewards. To define a ``large reward'', a comparison is made with a standard reward known as a reference reward.

The objective of the proposed policy is similar to the reinforcement comparison technique. However, choosing preferred actions is based on the visitor count to the states affected by these actions. That is, the most visited states indicate preferred actions in detriment to actions that lead to states with a lower number of visits. When the preference measurement for actions is known, the probability of action selection can be determined as follows:

$$\pi_t(a) = P_r\left\{a_t = a\right\} = \frac{e^{p_{t-1}(a)}}{\sum_b e^{p_{t-1}(b)}} \tag{14}$$

where $\pi_t(a)$ denotes the probability of selecting action a at step t and $p_t(a)$ is the preference for action a at time t. This is calculated by:

$$p_{t+1}(a_t) = p_t(a_t) + \beta(N_v(s,a_t) / N_{Ep}) \tag{15}$$

where s is the state affected by selecting action a at step t, $N_v(s, a_t)$ is the number of visits to state s, NEp is the total number of episodes and $\beta \in [0,1]$ is the control parameter that considers the influence level of preferred actions.

An experiment was carried out to compare the policies tested using *10* instances of *TSP* available in *TSPLIB* library (TSPLIB, 2010) and the results for the three policies tested simultaneously considering the value of the function and processing time, the *adaptive ε-greedy* policy had the best performance. This policy will be used in the version of the Q-learning algorithm implemented in this work.

3.4 The GRASP learning method

In GRASP metaheuristics the exploration and exploitation processes occur at different moments. The constructive phase explores the space of viable solutions and the local search improves the solution constructed in the initial phase of exploiting the area. Despite the clear delimitation of roles, the two phases of GRASP metaheuristics work in collaboration. The good performance of local search algorithm varies with the neighbourhood chosen and depends substantially on the quality of the initial solution. (Feo & Resende, 1995).

This section presents a hybrid method using Q-learning algorithm in the constructive phase of GRASP metaheuristics. In traditional GRASP metaheuristics iteration is independent, in other words, actual iteration does not use information obtained in past iterations (Fleurent & Glover, 1999). The basic idea of this proposed method is to use the information from the Q-values matrix as a form of memory that enables good decisions to be made in previous iterations and avoids uninteresting ones. This facilitates the exploration and exploitation process.

Based on the results of using an isolated Q-learning algorithm to solve the small instances of *TSP*, the approach proposed may significantly improve metaheuristic performance in locating the global optimum.

In the GRASP-Learning method, the Q-learning algorithm is used to construct initial solutions for GRASP metaheuristic. A good quality viable solution is constructed at each iteration of the algorithm using information from the Q-values matrix. The Q-values matrix can then be used at each GRASP iteration as a form of adaptive memory to allow the past experience to be used in the future. The use of adaptive word means that at each iteration new information is inserted by using the matrix Q. This influences the constructive phase of the next iteration.

The Q-learning algorithm will therefore be implemented as a randomized greedy algorithm. The control between "greed" (Exploration) and "randomness" (Exploitation) is achieved using the parameter ε of the transition rule defined in equation *13*. The reward matrix is determined as follows:

$$R(s,a) = \frac{1}{d_{ij}} * N_v(s,a) \tag{16}$$

where $1/d_{ij}$ is the inverse distance between the two cities c_i and c_j (city c_i is represented by state s and city c_j by the state accessed by choosing action a), and $N_v(s,a)$ is the number of visits to the current state.

The procedure of the Q-learning algorithm proposed for the GRASP constructive phase applied to the symmetric TSP is:

A table of state-action values Q (Q-values matrix) initially receives a zero value for all items. An index of table Q is randomly selected to start the updating process and the index then becomes state s0 for Q-learning. State s_1 can be obtained from state s_0 using the *ε-greedy* adaptive transition rule of the following way:

- Randomly, selecting a new city of the route, or
- Using the maximum argument Q in relation to the previous state s_0.

Given states s_0 and s_1, the iteration that updates table Q begins using the equation *4*. The possibility of selecting a lower argument state through randomness ensures the method ability to explore other search areas. Table Q is obtained after a maximum number of

episodes. The route to *TSP* is taken from this matrix. Constructing a solution for *TSP* after obtaining matrix *Q* is achieved as follows:

- Copy the data of matrix *Q* to auxiliary matrix Q_1;
- Select an index *l* that shows an initial line of table Q_1 (corresponding to the city at the start of the route)
- Starting from line *l* choose the line's greatest value, index *c* of this value is the column of the next city on the route.
- Attribute a null value to all the values of column *c* at Q_1, to ensure there is no repetition of cities on the route, repeating the basic restriction of *TSP*;
- Continue the process while the route is incomplete.

At each GRASP iteration the matrix *Q* is updated by executing the Q-learning algorithm. In other words, at the end of *Nmax* iterations Q-learning algorithm will have executed *Nmax***NEp* episodes, where *NEp* denotes the number of episodes executed for each iteration. Updating the Q-values matrix at each GRASP iteration improves the information quality collected throughout the search process. Fig. 7, shows a overview of the GRASP-Learning method.

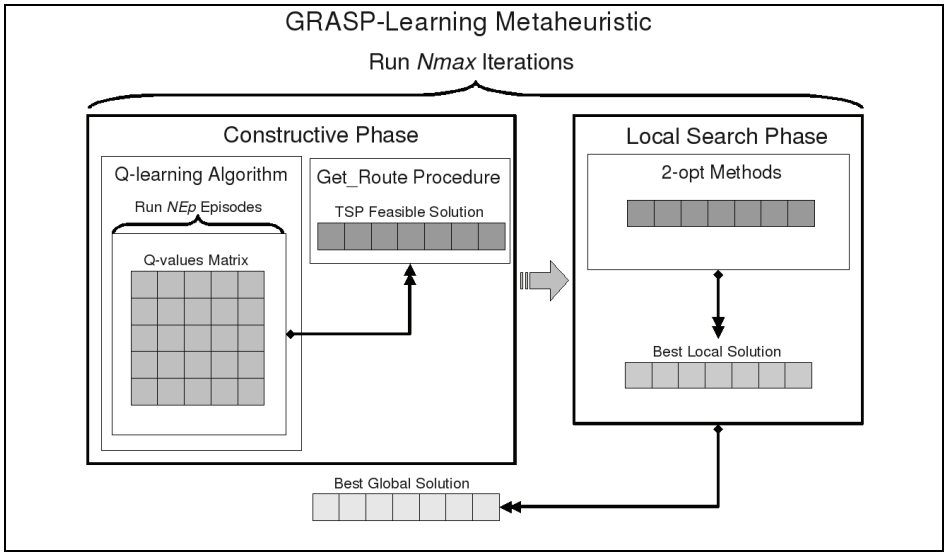


Fig. 7. Framework of the GRASP-Learning Method.

The local search phase in GRASP-Learning suffers no modifications. That is, it uses the same traditional GRASP algorithm that was implemented in this study as a descending method *2-Opt*. This local search technique only moves to a neighbouring solution if it improves in objective function. The Fig. 8, presents a simplified form of the pseudo code method GRASP-learning proposed.

When examining the pseudo code GRASP-Learning algorithm, significant modifications are seen in lines *3* and *5*. Function *MakeReward* in line *3* uses the distance matrix *D* to create a reward matrix, based on Equation *16*. In line *5* the function *Qlearning* returns a viable solution *S* for *TSP* and the updated Q-values matrix *Q*, which is used in the next iteration. The other aspects of GRASP-learning algorithm are identical to traditional GRASP metaheuristics.

Algorithm 4 GRASP-Learning Algorithm

```
1: procedure GRASP-LEARN(D, α, ϵ, γ, Nmax)
2:     f(S*) ← +∞
3:     R ← MakeReward(D)
4:     while i ≤ Nmax do
5:         [S, Q] ← Qlearning(R, α, ϵ, γ, Q)
6:         S' ← LocalSearch(S, Viz(S))
7:         if f(S') < f(S*) then
8:             S* = S'
9:         end if
10:        i ← i + 1
11:    end while
12:    return S*                    ▷ Best Solution
13: end procedure
```

Fig. 8. Pseudo code of GRASP-Learning Algorithm.

3.5 The cooperative genetic-learning algorithm

The hybrid genetic algorithm proposed in this section uses reinforcement learning to construct better solutions to the symmetric traveling salesman problem. The method main focus is the use the Q-learning algorithm to assist the genetic operators with the difficult task of achieving balance between exploring and exploiting the problem solution space. The new method suggests using the Q-learning algorithm to create an initial population of high fitness with a good diversity rate that also cooperates with the genetic operators.

In the learning process, the Q-learning algorithm considers using either the knowledge already obtained or choosing new unexplored search spaces. In other words, it has the characteristics of a randomized greedy algorithm. It is greedy owing to its use of the maximum value of *Q(s, a)* to choose action *a*, contributing to a greater return in state *s*, and randomized since it uses an action choice policy that allows for occasional visits to the other states in the environment. This behaviour allows Q-learning to significantly improve the traditional genetic algorithm.

As mentioned previously, the initial population of the proposed genetic algorithm is created by Q-learning algorithm. This is achieved by executing Q-learning with a determinate number of episodes (*NEp*). *Tp* chromosomes are then created using the matrix *Q* (Q-values matrix), where *Tp* is the population size. The choice criteria in the creation of each chromosome is the value of *Q(s, a)* associated to each gene so that the genes with the highest value of *Q(s, a)* are predominantly chosen in chromosome composition[3].

In addition to creating the initial population, the Q-learning algorithm cooperates with the genetic operators as follows:

- In each new generation, the best solution *S** obtained by the genetic operators is used to update the matrix of Q-values *Q* produced by the Q-learning algorithm in previous generations.
- After updating matrix *Q*, it is used in subsequent generations by the genetic algorithm. This iteration process rewards the state-action pairs that compose the

[3] A chromosome is an individual in the population (for example, a route to *TSP*), whereas a gene is each part that composes a chromosome (in *TSP* a city composes a route)

best solutions from the last generations. They receive an additional increment that identifies them as good decisions. Calculating the incremental value is based on the following theory:

- According to the model of *TSP* described in section 3.1, a solution for the model can be denoted by Fig. 9.

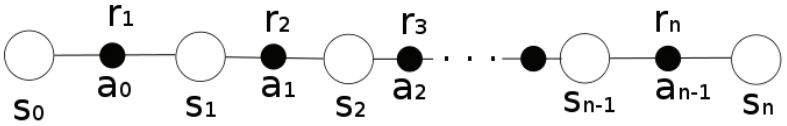


Fig. 9. *TSP* solution, modelled as a reinforcement learning Problem.

In the Fig. 9, the sequence of states, actions and rewards corresponds to a solution S constructed during an episode of Q-learning algorithm, where s_i, $i=0,\ldots,n$ is the current state and r_j, $j=1,\ldots,n$ is the immediate reward received by executing change s_i, for s_{i+1}. In this scenario, with solution S^*, the accumulated reward value R_i is easily calculated:

$$\begin{aligned} Q(s_0, s_1) &= r_1 + r_2 + r_3 + \ldots + r_{n-1} + r_n = R_0 \\ Q(s_1, s_2) &= r_2 + r_3 + \ldots + r_{n-1} + r_n = R_1 \\ &\vdots \\ Q(s_{n-1}, s_n) &= r_n = R_n \end{aligned} \tag{17}$$

- As mentioned in section 2.3, the Q-learning algorithm uses equation 4 in its updating process. However, since the value of R_i for solution S^* is already known, the incremental value proposed here can be calculated using:

$$Q(s_i, a_i) = Q(s_i, a_i) + \theta\left[R_i - Q(s_i, a_i)\right] \tag{18}$$

where a_i is the action of leaving state s_i and choosing to go to state s_{i+1}, and θ is a parameter that considers the importance of incremental value based on the number of visits to each state, denoted by:

$$\theta = N_v(s_i, a_i) / NEp \tag{19}$$

where, N_v is the number of visits to state-action pair (s_i, a_i), e NEp represents the number of episodes parameter of the Q-learning algorithm.

Fig. 10, shows a overview of the cooperative Genetic-learning method used.

The genetic operators were implemented identically to the traditional genetic algorithm, using a "dependent" spinner for selection. This allows each individual to be chosen proportionally to their value in the fitness function. The crossover of two points was used for the crossover operator. The mutation operator consists of a change in position between two cities on a route. The pseudo code shown in the Fig. 11 summarizes the method described.

The changes proposed in the Cooperative Genetic-learning algorithm that modify the traditional genetic algorithm can be seen in the pseudo code of lines *3, 7, 8, 9* and *15*. In line *3* the function *GeneratePop* executes Q-learning algorithm and creates the initial population.

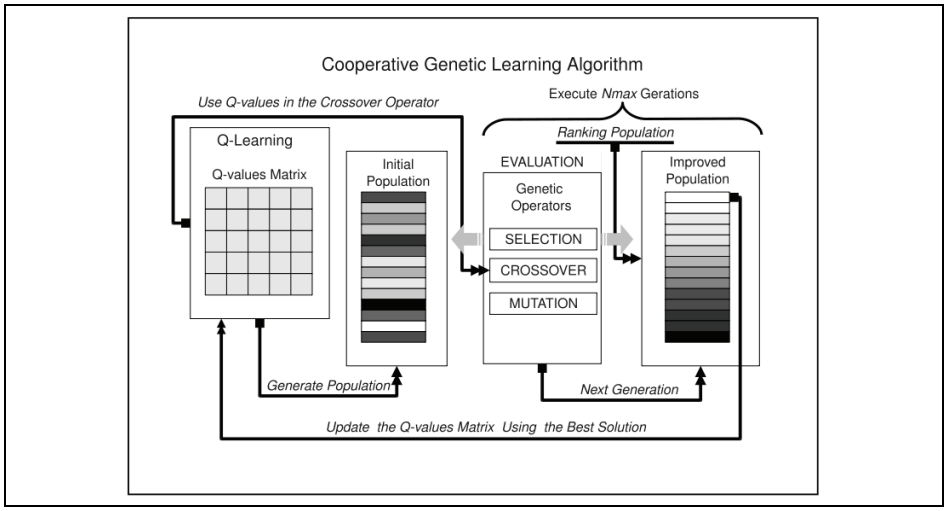


Fig. 10. Framework of the Cooperative Genetic-learning.

Algorithm 5 Cooperative Genetic-learning Algorithm

```
1:  procedure CG-LEARN(Tc, Tm, Tp, MaxG, L)
2:      Q ← QLearning(R, α_q, ε, γ, NEp)
3:      Pop ← GeneratePop(Q, Tp)
4:      Pop ← evaluate(Pop)
5:      for i = 1...MaxG do
6:          Pop ← Select(Pop)
7:          if Div(Pop) < L then
8:              CQ ← GetCromo(Q, 1)
9:              Pop ← Crossover(Pop, CQ, Tc)
10:         else
11:             Pop ← Crossover(Pop, Tc)
12:         end if
13:         Pop ← Mutation(Pop, Tm)
14:         Pop ← SortPop(Pop)
15:         Q ← UpdateQ(Q, Pop[1])
16:     end for
17:     return Best solution Pop
18: end procedure
```

Fig. 11. Pseudo code of Cooperative Genetic-learning Algorithm

In line *7* the diversification rate of the population is compared with limit *L*. If the limit *L* is lower (less than *20%*), the crossover operator will use a chromosome from an extra execution of the Q-learning algorithm (line 9). In line 15 the matrix of Q-values Q is updated using information from the best solution of the current population.

The hybrid methods described in this section are modified versions of GRASP metaheuristics and genetic algorithms. Modifications used the Q-learning algorithm as an

intelligent strategy for exploration and/or exploitation of space solutions of the symmetric traveling salesman problem.
In GRASP metaheuristics, modifications use Q-learning algorithm to construct its initial solutions. According to (Feo & Resende, 1995), one of the disadvantages of traditional GRASP metaheuristics is the independence of its iterations. That is, the algorithm does not keep information on the history of solutions found in past iterations. These authors cite the good quality of the initial solution, as one of the factors that contribute to the success of a local search algorithm. The proposed use of Q-learning algorithm as an exploratory starting point for the GRASP metaheuristic, as described here, overcomes both deficiencies of the traditional method cited by (Feo & Resende, 1995). This is owing to the fact that Q-learning algorithm can produce good quality initial solutions and uses its Q-values matrix as a form of adaptive memory, allowing good decisions made in the past to be repeated in the future.
In relation to the genetic algorithm, the proposed method (cooperative Genetic-learning) used the Q-learning algorithm to create an initial population. A good quality initial population was achieved, both in the value of its objective function and diversity level. Another important innovation proposed in this method was the cooperation between Q-learning algorithm and the genetic operators.
Tests were carried out with two distinct versions of the genetic hybrid algorithm to determine the importance of the proposed cooperation in the algorithm. One test used the cooperative process and the other did not. The results are presented in the next section.

4. Experimental results

Before presenting the computational results it is important to note that the experiments carried out do not aim to find an optimum solution for *TSP*. Their objective is to compare the performance of traditional methods with the new methods proposed. There was no concern regarding the quality of entrance parameters during the experiments. All the algorithms were executed under the same condition, that is, the same number of iterations and identical values for the common adjustable parameters were used.
Tests were carried out using 10 instances of the *TSPLIB* library (TSPLIB, 2010), on a desktop computer with: 2.8 *Ghz*, 2 Gb of RAM memory and Linux operating system. Processing time was measured in seconds. Since the algorithms are nondeterministic, the results are a mean of *30* executions for each instance.
The information about instances used in the experiment is presented in Table 1:

Instance Name	Description	Best value
gr17	City problem (Groetschel)	2085.00
bays29	Bavaria (street distance)	2020.00
swiss42	Switzerland (Fricker)	1273.00
gr48	City problem (Groetschel)	5046.00
berlin52	Locations in Berlin (Germany)	7542.00
pr76	City problem (Padberg/Rinaldi)	108158.00
gr120	Germany (Groetschel)	6942.00
ch150	City problem (Churritz)	6528.00
si175	Vertex TSP (M. Hofmeister)	21407.00
a280	Drilling problem (Ludwig)	2579.00

Table 1. Information about TSPLIB instances utilized.

4.1 Comparison of the performance of the GRASP metaheuristics implemented

This section compares the results obtained with the computational implementation of GRASP metaheuristics, reactive GRASP and the new method proposed GRASP-Learning. All the algorithms were executed under the same parameter conditions. The values of adjustable parameters are listed in Table 2.

Instance	All	GRASP	Reactive GRASP	GRASP-Learning			
TSPLIB	Number Iterations	α	α	Episodes	α_q	γ	ε
gr17	300	0.8	*	10	0.9	1.0	*
bays29	300	0.8	*	10	0.9	1.0	*
swiss42	300	0.8	*	20	0.9	1.0	*
gr48	300	0.8	*	20	0.9	1.0	*
berlin52	300	0.8	*	50	0.9	1.0	*
pr76	300	0.8	*	100	0.9	1.0	*
gr120	300	0.8	*	100	0.9	1.0	*
ch150	300	0.8	*	150	0.9	1.0	*
si175	300	0.8	*	200	0.9	1.0	*
a280	300	0.8	*	200	0.9	1.0	*

Table 2. Adjustable parameters for the metaheuristics GRASP (*Adaptive parameters)

When examining Table 2, it is important to understand the parameters α of reactive GRASP and ε of GRASP-Learning. Both are self-adjustable values that vary at interval [*0, 1*]. Another important observation is that the parameters α (traditional and reactive version) are not equal to the parameter α_q of GRASP-Learning. In other words, parameter α is used to control the indices of "greed" and randomness in traditional GRASP and reactive GRASP. Parameter α_q is the coefficient of learning Q-learning algorithm used in the hybrid GRASP version.

The values listed in Table 3 correspond to the mean of *30* executions obtained with *10* instances of the symmetric traveling salesman (objective function value, the processing time in seconds). Fig. 12 and Fig. 13 compare the three versions of metaheuristics used, considering the objective function value and processing time, respectively. Data shown in graphs were normalized to improve their visualization and understanding.

The objective function values were normalized by the known optimum value at each instance of *TSPLIB* and the processing time was normalized by the mean execution time of each metaheuristic for each instance, this is calculated by Equation *20*.

$$M_i = \left(\sum_{j=1}^{m} T_{ij} \right) / m \;\; \forall i = 1,2,\ldots,n$$

$$TN_{ij} = T_{ij} / M_i \tag{20}$$

where, n is the number of instances used in the experiment, m is the number of algorithms tested. T_{ij} is the processing time for instance i executing algorithm j, and TN_{ij} is the value of normalized time for instance i executing algorithm j.

TSPLIB Instance	TSPLIB Optimal	Traditional GRASP		Reactive GRASP		GRASP Learning	
Name	Value	Value	Time	Value	Time	Value	Time
gr17	2085.00	2085.00	25.30	2085.00	21.07	2085.00	17.03
bays29	2020.00	2085.63	112.41	2060.50	103.96	2030.30	46.34
swiss42	1273.00	1441.43	373.62	1385.07	354.05	1281.40	84.64
gr48	5046.00	5801.77	559.81	5454.17	532.62	5442.77	127.41
berlin52	7542.00	8780.73	752.37	8420.93	599.10	8053.60	156.36
pr76	108159.00	140496.00	1331.51	131380.33	1724.84	129707.33	719.65
gr120	6942.00	10553.61	5000.10	8773.33	5945.75	8540.47	2443.82
Ch150	6528.00	10785.59	11167.91	9304.40	12348.96	7012.93	1753.29
si175	21407.00	25733.07	21509.92	23646.14	12803.00	22700.35	8830.08
a280	2579.00	5799.77	40484.48	4244.19	37075.25	2991.30	8442.40

Table 3. Performance of GRASP, reactive GRASP and GRASP-Learning Metaheuristics

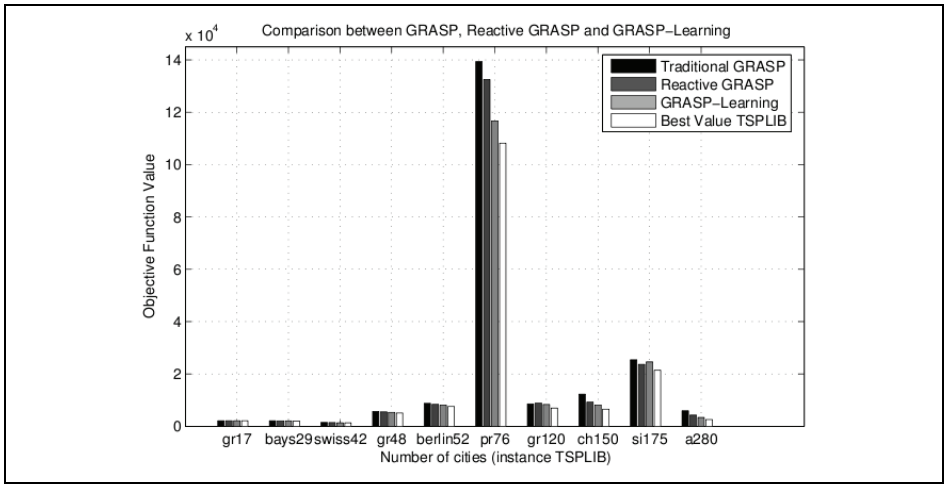


Fig. 12. Traditional GRASP, Reactive GRASP and GRASP-learning (Objective Function)

Upon analysis, the results in Table 6 show that the metaheuristic GRASP-Learning had better mean results than the traditional GRASP metaheuristic. It even performed better than reactive GRASP, the improved version of traditional GRASP metaheuristics. The results for objective function achieved with the hybrid method were better than those achieved using the traditional method. In the worst case, a tie occurred since all versions found the optimum instance *gr17*. In the best case, the new method showed a cost reduction of *48.42 %* (instance *a280*).

The hybrid method achieved very competitive results when compared to reactive GRASP. It performed better in most instances for objective function values and decreased the cost of instance *a280* by *29.52%*.

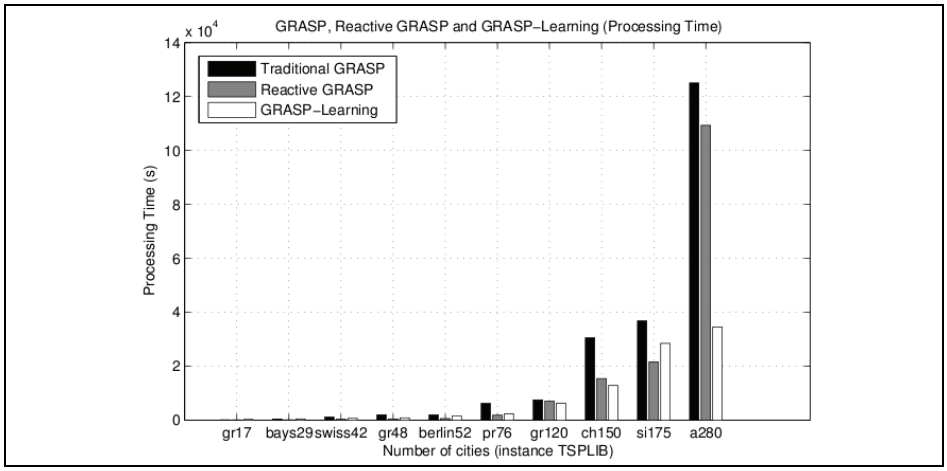


Fig. 13. Traditional GRASP, Reactive GRASP and GRASP-learning (Processing Time)

When comparing the processing time results, the advantage of the GRASP-Learning metaheuristic is even clearer. When compared with traditional GRASP, it achieved a decrease in processing time of *32.69%* for instance *gr17* (worst case) and *84.30%* for instance *ch150* (best case). In comparison with reactive GRASP, the hybrid method achieved *19.17%* for instance *gr17* in the worst case and *85.80* in the best case (instance *ch150*).

The best processing time results using the GRASP-Learning metaheuristic were achieved owing to the good quality of initial solutions created in the constructive phase of Q-learning algorithm. Since the metaheuristic started with good quality solutions, it was able to accelerate the local search. The GRASP-Learning metaheuristic also has the advantage of using the memory between iterations. This means that the quality of the initial solution is improved at each iteration, based on the information in the Q-values matrix, which is updated at each iteration by the Q-learning algorithm. It is important to note that good results are expressed as the instances grow, since greater the instance of *TSP*, the greater the difficulty level of a solution. Therefore, the advantage of Q-learning algorithm to generate good initial solutions is a consequence of the use of the matrix of q-values as a mechanism of adaptive memory.

4.2 Performance comparison of the genetic algorithms implemented

The experimental results for all versions of the genetic algorithms used were achieved using the same entrance parameters. These values are shown in Table 4.

The Table 5 shows the mean obtained with *30* executions for each version of the genetic algorithms.

The charts in the Fig. 14 and Fig. 15 compare the three versions of algorithms tested (traditional genetic, Genetic-learning, cooperative Genetic-learning) considering objective function value and processing time, respectively. Graph data were normalized in a similar form to the process used in GRASP metaheuristics. That is, the objective function value was normalized by the optimum value of each instance of *TSPLIB* and the processing time was normalized by the mean execution time of each algorithm for each instance, as described in Equation *20*.

TSPLIB Instance	All Genetic Algorithms				Genetic Learning Algorithm			
Name	Number of Generations	Tc	Tm	Tp	Number of Episodes	α_q	γ	ε
gr17	1000	0.7	0.2	100	500	0.8	1	*
bays29	1000	0.7	0.2	100	500	0.8	1	*
swiss42	1000	0.7	0.2	100	500	0.8	1	*
gr48	1000	0.7	0.2	100	500	0.8	1	*
berlin52	1000	0.7	0.2	100	500	0.8	1	*
pr76	1500	0.7	0.2	100	1000	0.8	1	*
gr120	2000	0.7	0.2	100	1000	0.8	1	*
ch150	2000	0.7	0.2	100	2000	0.8	1	*
si175	2000	0.7	0.2	100	2000	0.8	1	*
a280	2000	0.7	0.2	100	2000	0.8	1	*

Table 4. Adjustable parameters for the Genetics algorithms (*Adaptive parameters)

TSPLIB Instance	TSPLIB Optimal	Traditional Genetic Algorithm		Genetic Learning Algorithm		Cooperative Genetic Learning Algorithm	
Name	Value	Value	Time	Value	Time	Value	Time
gr17	2085.00	2104.97	48.73	2087.37	48.03	2085.00	51.77
bays29	2020.00	2286.23	52.27	2252.13	52.24	2166.47	57.45
swiss42	1273.00	1614.20	54.52	1546.53	55.01	1457.73	63.66
gr48	5046.00	6839.77	55.99	5967.90	56.19	5744.90	66.59
berlin52	7542.00	10095.63	55.55	8821.93	55.68	8679.43	69.17
pr76	108159.00	189659.00	89.49	165281.67	95.36	132100.33	123.60
gr120	6942.00	16480.73	132.05	12205.87	140.41	8787.00	211.51
ch150	6528.00	19705.47	142.44	9612.33	153.26	8803.37	247.47
si175	21407.00	30860.97	151.86	29264.13	193.25	24818.03	285.86
a280	2579.00	13642.07	205.92	3852.67	274.55	3768.03	543.17

Table 5. Performance of Genetic, Genetic Learning and Cooperative Genetic-Learning.

When analyzing the experimental results, the Genetic-learning algorithms achieved better results for objective function value but not for processing time.

The result for objective function value is achieved owing to the good quality of the initial population generated by Q-learning algorithm and cooperative iteration of genetic operators with the Q-values matrix. The good performance of the objective function value is mainly noted as the instances grow. For example, when comparing the traditional genetic algorithm with cooperative Genetic-Learning improvement in the worst case is 0.95% (instance gr17) and in the best case 72.38% (instance a280).

The processing time of cooperative Genetic-learning algorithm is at a disadvantage to the other two versions since this version uses Q-learning algorithm in initial population construction and cooperation with the genetic operators. The time spent processing episodes *NEp* of Q-learning algorithm is added to the final execution time.

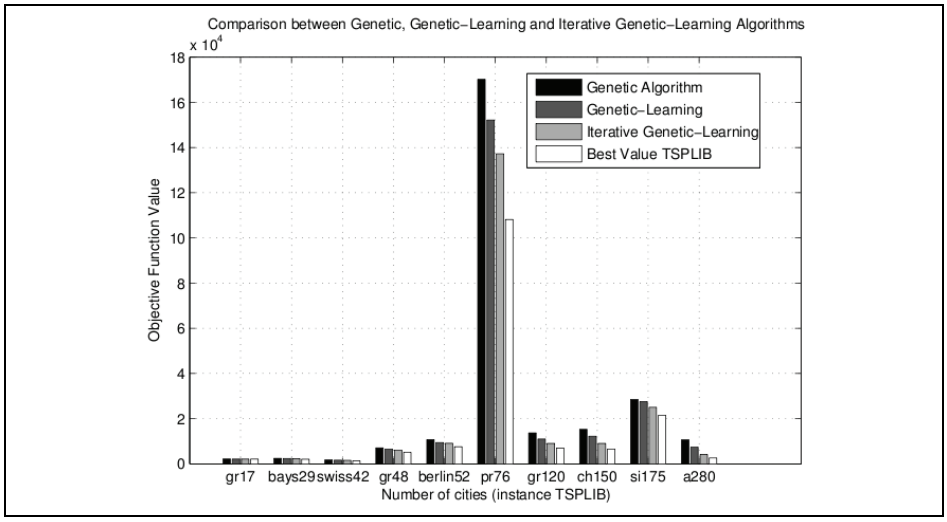


Fig. 14. Genetic, Genetic Learning and Cooperative Genetic-Learning. (Objective Function)

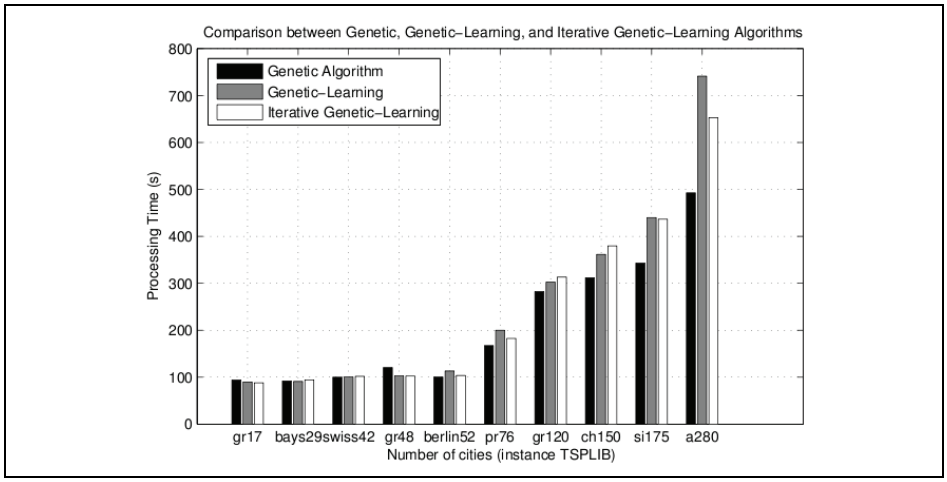


Fig. 15. Genetic, Genetic Learning and Cooperative Genetic-Learning. (Processing Time)

5. Conclusions

This section presents observations, conclusions and perspectives of this study. The text is subdivided into subsections according to the method proposed.

5.1 GRASP-learning method

The proposed method GRASP-Learning satisfies two requirements of GRASP metaheuristics: a good final performance based on good quality initial solutions and no memory mechanism between iterations.

Concerning the quality of initial solutions, using Q-learning algorithm to construct these olustions showed promising results for the objective function value and processing time.
The purpose of the randomized greedy Q-learning algorithm in the constructive phase of GRASP metaheuristic is not only to provide good initial solutions, but also a form of adaptive memory. This allows good decisions made in past iterations to be repeated in the future. The adaptive memory mechanism is implemented using information from the Q-values matrix created by Q-learning algorithm. The term adaptive memory is used since the Q-values matrix is updated at each episode of Q-learning. This incorporates the experience achieved by the learning agent in each update.
The GRASP-learning metaheuristic achieved better results than traditional GRASP and reactive GRASP when comparing general performance. In relation to processing time, good performance by GRASP-Learning is especially noted as the instances grow. This is confirmed by the fact that the hybrid method, traditional GRASP and reactive GRASP are only different in the constructive phase. Also, the partially greedy algorithm used in GRASP and reactive GRASP has complexity O(n2), while Q-learning algorithm has complexity O(NEp*n) where NEp is the number of episodes and n the instance size. Since updating the Q-values during execution of GRASP-learning is cumulative (in k iterations of algorithm GRASP-learning, $k*NEp$ episodes of Q-learning are executed), the algorithm can be parameterized with a relatively small value of NEp. Based on the complexity orders of the algorithms, Q-learning outperforms the partially greedy algorithm as the instances grow.
Another important aspect that confirms the lower processing time of the hybrid method is the good quality of initial solutions constructed by Q-learning algorithm, since the GRASP metaheuristic starting with good initial solutions had an accelerated local search.

5.2 Genetic learning algorithm

The hybrid genetic algorithm proposed in this study showed very significant results, mainly in its cooperative version. Updating the Q-values matrix with elite solutions from each population produced a significant improvement in performance, especially in objective function values. The traditional version achieved better processing time results. This was expected since the traditional version constructed a randomized initial population. Learning versions use Q-learning algorithm; therefore, the execution time of the episodes is added to their processing time. Despite the non-significant processing time results, the Genetic-learning algorithm appreciably improved the objective function value. This was already substantial when comparing the quality of initial populations (see Subsection 4.2) since the population created by Q-learning algorithm achieved better quality (better Fitness) and an equal diversification rate for larger instances.
This method also contributes through the mutual cooperation between Q-learning algorithm and the genetic operators that exchange information throughout the evolution process. This cooperation offers a range of possibilities for the parallel implementation of these algorithms, for example by using a cooperative/competitive strategy to solve the traveling salesman problem.

5.3 Future works

The methods proposed in this study are tested using only the symmetric traveling salesman problem. Although *TSP* is a classic combined optimization problem from which many other practical problems can be derived, applying the methods proposed here to other problems of this class requires careful modelling.

Another important factor for improvement concerns is the instance sizes in the test. For questions of speed in method validation, small and medium-size instances of T*SP* were used.
Based on the developed studies and possible improvements, prospects exist for the following future works:

- Computational tests with instances of TSP with a higher number of cities to determine the behaviour of the methods proposed when using larger instances.
- Applying GRASP-Learning metaheuristic and Genetic-learning algorithm to other combinatorial optimization problems.
- Using parallel hybrids for the traveling salesman problem based on the hybrid methods proposed here. This study is currently being developed in a PhD thesis of *PPgEEC/UFRN* (Queiroz, J.P. et al., 2008) with very interesting results.
- Investigating the use of the reinforcement learning Q-learning algorithm to improve other metaheuristics.

6. References

Feo T. & M. Resende (1995). Greedy randomized adaptive search procedures, *Journal of Global Optimization*, Vol. 06, (March 1995), Page numbers (109 – 133) ISBN : 0925 - 5001

Fleurent, C. & Glover, F. Improved constructive multistart strategies for the quadratic assignment problem using adaptive memory, I*nforms Journal on Computing* Vol. 11, No. 2, Spring 1999, pp. 198-204, DOI: 10.1287/ijoc.11.2.198.

Lima, F. C. & Melo J. D. & Doria Neto A. D. Using Q-learning Algorithm for Initialization of the GRASP metaheuristic and Genetic Algorithm, *International Joint Conference on Neural Networks (IJCNN), 2007, IEEE Proceedings of IJCNN 2007*, Florida, pages. 1243-1248, 2007.

M. Prais & C. Ribeiro(1998) Reactive GRASP: An application to a matrix decomposition problem in TDMA traffic assignment. Catholic University of Rio de Janeiro, Department of Computer Science, Rio de Janeiro, 1998.

Puterman M. L. Markov Decision Processes Discrete Stochastic Dynamic Programming, John Wiley e Sons, Inc, New York, USA, 1994.

Queiroz, J. P. & Lima, F.C. & Magalhaes R.M. & Melo, J. D. & Adriao, Doria Neto A. D. A parallel hybrid implementation using genetic algorithm, GRASP and Reinforcement Learning, *International Joint Conference on Neural Networks*, pages 2798-2803, 2008.

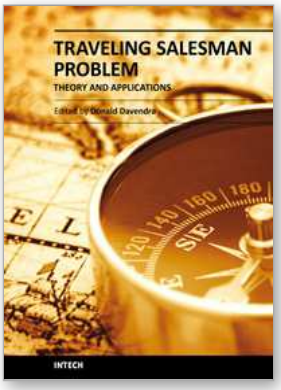
Ramos, I. C. O. & Goldbarg, M. C. & Goldbarg, E. F. G. & Doria Neto, A. D. & Farias, J. P. F. ProtoG Algorithm Applied to the Traveling Salesman Problem. In: *XXIII International Conference of the Chilean Computer Science Society: Computer Society, IEEE*, pages 23--30, Los Alamitos, 2003.

Randy, L. Haupt & Sue Ellen Haupt. *Pratical Genetic Algorithms*, Second edition, a John Wiley and Sons, Inc.,Publication, New Jersey, 2004.

Sutton, R.S. & Barto, A.G. *Reinforcement Learning: An Introduction*, MIT Press, Cambridge, MA,1998.

TSPLIB, on-line library,http://www.iwr.uni-heidelberg.de/groups/comopt/software/ TSPLIB95/, accessed April 2010.

Watkins, Christopher J. C. Hellaby. *Learning from delayed rewards*, PhD thesis, Cambridge University, Cambridge, England 1989.



Traveling Salesman Problem, Theory and Applications
Edited by Prof. Donald Davendra

ISBN 978-953-307-426-9
Hard cover, 298 pages
Publisher InTech
Published online 30, November, 2010
Published in print edition November, 2010

This book is a collection of current research in the application of evolutionary algorithms and other optimal algorithms to solving the TSP problem. It brings together researchers with applications in Artificial Immune Systems, Genetic Algorithms, Neural Networks and Differential Evolution Algorithm. Hybrid systems, like Fuzzy Maps, Chaotic Maps and Parallelized TSP are also presented. Most importantly, this book presents both theoretical as well as practical applications of TSP, which will be a vital tool for researchers and graduate entry students in the field of applied Mathematics, Computing Science and Engineering.

How to reference
In order to correctly reference this scholarly work, feel free to copy and paste the following:

Francisco Chagas De Lima Júnior, Adriao Duarte Doria Neto and Jorge Dantas De Melo (2010). Hybrid Metaheuristics Using Reinforcement Learning Applied to Salesman Traveling Problem, Traveling Salesman Problem, Theory and Applications, Prof. Donald Davendra (Ed.), ISBN: 978-953-307-426-9, InTech, Available from: http://www.intechopen.com/books/traveling-salesman-problem-theory-and-applications/hybrid-metaheuristics-using-reinforcement-learning-applied-to-salesman-traveling-problem-

INTECH
open science | open minds

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821