A Survey on Hybridizing Genetic Algorithm with Dynamic Programming for Solving the Traveling Salesman Problem

PHAM Dinh Thanh
Faculty of Mathematics - Physics - Informatic
Tay Bac University
SonLa, Vietnam
thanhpd05@gmail.com

HUYNH Thi Thanh Binh
School of Information and Communication Technology
HaNoi University of Science and Technology
HaNoi, Viet Nam
binhht@soict.hut.edu.vn

BUI Thu Lam
Faculty of Information Technology
Le Quy Don Technical University
HaNoi, Viet Nam
lam.bui07@gmail.com

Abstract— **Traveling Salesman Problem (TSP) is a well-known NP-hard problem. Many algorithms were developed to solve this problem and gave the nearly optimal solutions within reasonable time. This paper presents a survey about the combination Genetic Algorithm (GA) with Dynamic Programming (DP) for solving TSP. We also setup a combination between GA and DP for this problem and experimented on 7 Euclidean instances derived from TSP-lib. Experimental results are reported to show the efficiency of the experimented algorithm comparing to the genetic algorithm.**

Keywords-Traveling Salesman Problem, Genetic Algorithm, Dynamic Programming, Brand & Bound algorithm

I. INTRODUCTION

The traveling salesman problem is an important problem in computing fields and has many applications in the real-world such as scheduling, vehicle routing, *VLSI* layout design… The problem was first formulated in 1930 and became one of the most intensively studied problems in optimization. Until now, researchers have obtained many significant results for this problem.

This paper introduces overview of the hybridizing Genetic Algorithms with Dynamic Programming for Solving *TSP* and then presents a combination *GA* with *DP* (called *CGADP*) for solving this problem. In *CGADP*, the solutions found by genetic algorithm will be selected for applying a local search based on *DP*. We experimented *CGADP* on 7 Euclidean instances derived from *TSP*-lib [22] and compare the result with *GA* [5]. Experimental results show that the solutions found by *CGADP* are better than the ones found by *GA* [5] on both the min and the mean cost values. The convergence rate of the solution found by *CGADP* is faster than that of *GA*.

The rest of this paper is organized as follows. In section II, we present overview of *TSP*. Section III surveys the directions of the combination *GA* with *DP* for solving *TSP*. The details of our experiments and the computational and comparative results are given in section IV. The paper concludes with section V with some discussions on the future extension of this work.

II. OVERVIEW OF TSP

TSP is stated as following: Let 1, 2, …, *n* be the labels of the n cities and $C = [c_{i,j}]$ be an *n* x *n* cost matrix where $c_{i,j}$ denotes the cost of traveling from city i to city j. *TSP* is the problem of finding the *n*-city closed tour having the minimum cost such that each city is visited exactly once. The total cost *A* of a tour is.

$$A(n) = \sum_{i=1}^{n-1} c_{i,i+1} + c_{n,1} \qquad (1)$$

TSP is formulated as finding a permutation of *n* cities, which has the minimum cost. This problem is known to be *NP*-hard [2, 4, 5]. Many algorithms have been proposed to solve this problem [2, 3, 4, 5, 7, 10, 11, 12, 14, 15, 17]. There are two main approaches for solving *TSP*: exact and approximate.

Exact approaches are almost always based on Dynamic Programming, Branch and Bound, Integer Linear Programming…and all gave the optimal solutions for *TSP*. However, the algorithms basing on these approaches have exponential running time as M. Held and R. M. Karp [1] pointed out Dynamic Programming takes $O(n^2.2^n)$ running time. Hence, they can only solve *TSP* with small number of the vertices as algorithms using branch and bound method are only able to give solutions for 40 – 60 cities sets and ones using linear programming solve with maximum for 200 cities sets.

In an attempt to solve larger instances, especially in such the *NP*-hard problem, approximation approaches have been concerned by researchers in recent years. Many approximation approaches were proposed for solving *TSP* such as 2-opt, 3-opt [2], simulated annealing [3], tabu search [4]; nature based optimization algorithms and population based optimization algorithms: genetic algorithm [16, 19, 20], evolutionary computation [5], neural networks [6], *DNA* computing [9]; swarm optimization algorithms: ant colony optimization [7], bee colony optimization [8]. The algorithms basing on these approaches can solve large instances and give approximate solutions near to the optimal solution within reasonable time.

In addition to above original approximation approaches, there is a different one combining basic heuristic methods called meta-heuristics. In [18], the authors applied local search heuristics to *GA* for solving *TSP*. The local search method they used is 2-opt. They presented three crossover operators (PMX, OX, POS) and two mutation operators (IVM, EM), then combine 2-opt with one pair of crossover and mutation operator in turn. After experimenting their algorithms on kroA100, kroB100 and kroC100 instances, they found that the combination of two genetic operators (IVM and POS) with 2-opt gave better solutions than the others did for solving *TSP* problem. They also implemented this combination but with 3-opt instead of 2-opt and came to the conclusion that the combination with 3-opt gave better solutions but converged to global optimum in more time.

Also using local search, Bernd Freisleben et al. proposed Genetic Local Search (*GLS*) for the *TSP* [20]. Their algorithm used the idea of hill climber to develop local search in *GA*. Their experiment showed that *GLS* is more effective in terms of not only running time, but also cost than ones in [21].

Besides exact and approximate approaches, a different one that is the combination of these two approaches, in which the combination *GA* with *DP* is most popular and it will be introduced in the next section.

III. THE COMBINATION GA WITH DP FOR SOLVING TSP

Algorithms using *DP* can give optimal solutions, but their computation time is too large whereas *GA* is inverse. So by combination *GA* with *DP*, researchers try to keep these algorithms' advantages and reduce their defects so as to not only give optimal solutions, but also shorten the running time. There are two popular ways of this combination to solve *TSP*: create new genetic operators using *DP* and to use *DP* as one step in *GA*.

A. Create new genetic operators using DP

By this way, *DP* algorithm is usually applied to crossover operators. Authors build *DP* functions or fitness functions based on *DP*.

In [16], Mutsunori Yagiura et al. proposed a new *GA* called *genetic DP*, in which they applied *DP* to create a new crossover operator. To implement this crossover, they defined a new concept called the partial order common of two candidate solutions (σ_1, σ_2). Its definition is as following: The partial order common of (σ_1, σ_2) is a set (denotes *D*) of the cities pairs (*i*, *j*) such that the city *i* is visited before the city *j* in both σ_1 and σ_2.

$$D = \{(i,j) \mid \sigma_1^{-1}(i) \le \sigma_1^{-1}(j) \text{ and } \sigma_2^{-1}(i) \le \sigma_2^{-1}(j)\} \qquad (2)$$

Then they argued that good solutions usually have a lot of common structures. Basing on this argument, they created their crossover operator, in which children σ are born such that the number of pairs (*i*, *j*) $(with\,(i,j) \in D)$ in σ is maximum. Hence, with supposing that a tour always starts from vertex 1, they built a *DP* formula to find the best child (which has minimum cost) of each pair of parents (formula (3)).

Basing on (3), the idea of crossover operator can be presented as follows: after choosing a pair of parents (σ_1, σ_2) randomly, they build a set *D* by (2). Then, the child σ_{12} starting from the vertex 1 will add the next vertex *j*. So, the children created by this crossover operator are always the best as possible. *Genetic DP*, therefore, is expected to find better solutions for this problem than the traditional genetic.

$$\begin{aligned} & f^*(\{1\},1) = 0 \\ & f^*(S,i) = \min_{j \in I(S)-\{i\}} \{f^*(S-\{i\},j) + \mathrm{d}_{ij}\}\ with\ S \in V^*(D) \\ & f^*(V) = \min_{i \in I(\mathrm{V})} \{f^*(V,i) + d_{ij}\} \\ & V^*(D) = \{S \subseteq V \mid j \in S \text{ and } (i,j) \in D \Rightarrow i \in S\} \\ & I(S) = \{i \in S \mid \text{ no } j \in S \text{ satisfies } j \ne j \text{ and } (i,j) \in D\} \end{aligned} \qquad (3)$$

Besides, *genetic DP* also creates a new mutation operator. However, this mutation can be used or not. The mutation is implemented by randomly perturbing the common partial order *D* as follows: choose a pair $i, j \in D$ such that $\sigma_1^{-1}(i) < \sigma_1^{-1}(\mathrm{j})$, and relax *D* by one of the following two operations:

$$\begin{aligned} & D := D - \{(i,k) \mid \sigma_1^{-1}(i) < \sigma_1^{-1}(k) \le \sigma_1^{-1}(j)\}, \\ & D := D - \{(k,j) \mid \sigma_1^{-1}(i) < \sigma_1^{-1}(k) \le \sigma_1^{-1}(j)\}. \end{aligned} \qquad (4)$$

Then they apply *DP* formula on the new *D* to obtain a new solution.

To show the efficiency of *Genetic DP*, the authors [16] experimented on 15 randomized Euclidean instances (5 instances for each *n* = 100, 200, 500). The experimental results showed that solutions found by *Genetic DP* were better than ones found by Multi-Local, Genetic-Local and Or-opt when sufficient computational time was allowed. But comparing with Lin-Kernighan algorithm [17], *Genetic DP* is worse in term of not only solutions quality, but also running time.

The authors [23] also applied *DP* to crossover operator in *GA* for solving *TSP*, but the difference is that they used Hill-Climbing (*HC*) instead of building a *DP* formula as in [16]. This *HC* method name is "Simplex", so their crossover operator was called "Simplex crossover".

The main idea of "Simplex" is that from an initial simplex with (*n*+1) points (where n is the number of cities) in a *n*-dimensional space, implement a sequence of elementary geometric transformations including reflection, contraction and extension such that the simplex adapts to the function landscape and finally surrounds the optimum. Hence, they implement "Simplex crossover" as following: find the best (x_{max}) and the worst (x_{min}) in the selected (*n*+1) individuals, then calculate by (5) to find the centroid (x_c) of these (*n*+1) individuals.

$$x_c = \frac{\sum_{i \ne \min}^{n+1} x_i}{n} \qquad (5)$$

After, they calculated "reflected point" (x_r), "expanded point" (x_e), "contracted point" (x_i), "contracted point towards the best one" (x_o) by (6).

$$\begin{aligned} x_r &= x_c + (x_c - x_{\min}) \\ x_e &= x_r + (x_r - x_{\min}) \\ x_i &= (x_{\min} + x_c)/2 \\ x_o &= (x_{\min} + x_{\max})/2 \end{aligned} \qquad (6)$$

Based on these calculated value, one of three elementary geometric transformations is implemented.

The results of hybrid method have high reliability and computing time is approximately 5 times smaller than GA's time. With larger test sets this hybrid method still maintains high reliability level, whereas the performance of the other methods decreases significantly.

B. *Using DP as a stage in GA*

Instead of using *DP* to intervene genetic operators as presented in the previous section, this section presents the ways to use *DP* as a preprocess step before implementing genetic operators.

In [23], addition to create a new genetic operator using *HC* as presented in subsection A of section III, Renders et al. also used *HC* to optimize the parents before applying crossover and mutation operators.

By this combination, they proposed *GHL* (Genetic algorithms hybridized with a Hill-climbing method following a Lamarckian). In the *GHL*, the solutions found by *HC* are used for performing crossover and mutation operator. And when evaluating the fitness of each individual, *GA* uses the results of *HC* working with an initial guess corresponding to this individual.

This hybrid method gave not only better solutions but also faster than *GA*.

The other combination between *GA* and *DP* is used for solving the single-vehicle pickup and delivery problem [24], in which, the *DP* is implemented before *GA* and used to find the optimal routing. The model of the hybrid genetic algorithms is showed in figure 1.

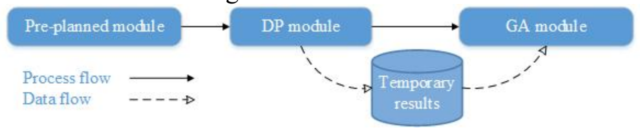


Figure 1. Model of hybrid GA

Figure 1 depicts the model of the hybrid genetic algorithm with *DP*, where a pre-planned module will arrange the tasks and prepare the information for the *DP* module. The *DP* module performs a dynamic programming algorithm. When a specific time is expired, the *DP* module will move its unfinished sub-routes to a temporary result pool; these unfinished sub-routes will be the initial population of genetic algorithm.

In [24] the authors also modified the objective function to accomplish a route in real time. A modified objective function is defined as follows:

$$Z_{dynamic} = Z_{objective} + [\alpha_1 \sum_{r \in V^+ \cup V^-} f_{delay}(r) + \alpha_2 \sum_{r \in V^+ \cup V^-} f_{overload}(r)] \quad (7)$$

The objective function Z*dynamic* is based on the objective function of the static case, but with some penalties. For any, $r \in V^+ \cup V^-$, the function *f*

the function $f_{delay}(r)$ and $f_{overload}(r)$ represent the vehicle delay time and overload at location r respectively where α_1 and α_2 are penalty coefficients.

The function $f_{delay}(r)$ and $f_{overload}(r)$ are defined as following (any $r \in V^+ \cup V^-$):

$$f_{delay}(r) = \begin{cases} t_r - b_r, \text{ if a vehicle arrives at location } r \text{ lately;} \\ 0, \text{ otherwise.} \end{cases}$$
$$f_{overload}(r) = \begin{cases} l_r - Q, \text{ if the current load } l_r, \\ \qquad \text{exceeds the vehicle capacity Q;} \\ 0, \text{ otherwise.} \end{cases} \quad (8)$$

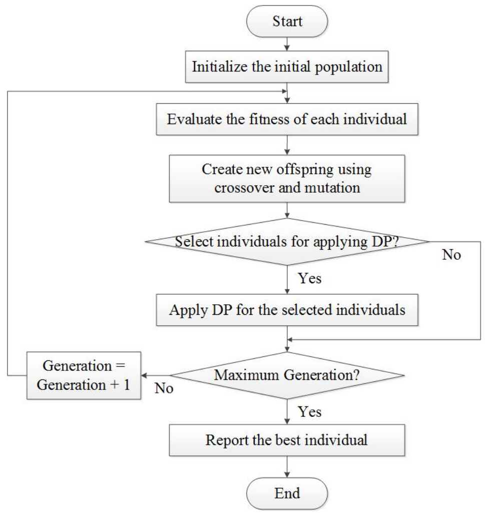
In experiments, the authors tested the hybrid genetic algorithm on five problems consisting of 10, 20, 30, 40 and 50 tasks and distance is Euclidean distance. The results show that, the proposed algorithm always obtains the lower solution cost than the traditional *GA*. Further, the proposed algorithm can generate incrementally better solutions at any time.

As previously mentioned, *DP* method found the best solution for *TSP*. However this approach has a large computational complexity and it is only suitable to small instances. Besides, *GA* can solve *TSP* problem with large instances [5, 13, 19, 20], but in some cases, *GA* gave worse solutions or slow convergence. So, in this paper, we present the combination of *DP* and *GA* [5]. This algorithm will be presented in the next section.

IV. CASE STUDIES

A. *The combination of genetic algorithm and dynamic programming*

This section presents the combination of genetic algorithm and dynamic programming (*CGADP*) for solving *TSP*, in which genetic algorithm was presented in [5].



Start
Initialize the initial population
Evaluate the fitness of each individual
Create new offspring using crossover and mutation
Select individuals for applying DP?
No
Yes
Apply DP for the selected individuals
Generation = Generation + 1
No
Maximum Generation?
Yes
Report the best individual
End

Figure 2. The combination of genetic algorithm and dynamic programming.

Figure 2 shows the flowchart of *CGADP*. This algorithm starts with randomly initializing the population in *GA*. We improve the *GA* for solving *TSP* by using local search. After each generation, it selects p_{dp}% individuals of the current population randomly and uses a local search procedure based on *DP* (in subsection B of section III) to generate new better individuals. The algorithm stops after the fixed number of generations.

Sketch of the *CGADP* algorithm is presented below:

```
Algorithm 1: CGADP
Input: The initial population P
Output: The optimization population P'
1.  Begin
2.    Evaluate the fitness of individuals in P
3.    i ← 0
4.    P_i ← P
5.    ng ← maximum number of generations
6.    While i < ng do
7.      ps ← size of the population
8.      for j := 1 to ps/2 do
9.        do crossover and mutation
10.       Insert offspring into P_{i+1}
11.     end for
12.     ndp ← number of individuals for applying DP
13.     l ← length of a segment
14.     for j := 1 to ndp do
15.       c ← select a random individual from P_{i+1}
16.       k ← random gen position
17.       c' ← apply DP(c, k, l);
18.       replace c by c' in P_{i+1}
19.     end for
20.     i ← i+1
21.   end while
22.   P' ← P_i
23.   return P'
```

B. Local Search using Dynamic Programming

Dynamic programming is a method which solves complex problems by decomposing them into simpler subproblems. There are many algorithms that apply *DP*. In this paper, we use the combination of local search using *DP* and *GA* for solving *TSP*.

DP is applied to find a better segment to replace in each tour. The main idea is described as following:

- Choose an individual and its gen randomly.
- Use *DP* to find the optimal value on the segment with a given length *l* beginning at the just chosen tour.

The cost function of a segment from *k*-th city to (*k*+*l*-1)-th city is calculated by:

$$C' = \sum_{i=k-1}^{k+l-1} c_{\pi_i,\pi_{i+1}} \qquad (9)$$

c_{π_i,π_j} : cost of traveling from city π_i to π_j.

k: a random gen position.
l: the length of segment (sub-tour).
$\pi_1, \pi_2, \ldots, \pi_n$: the permutation of cities 1,…, *n*.

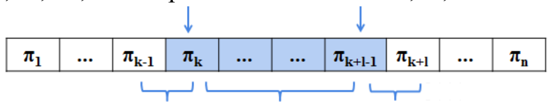


Figure 3. Calculation the cost of segment from *k*-th city to (*k*+*l*-1)-th city

When each city in the segment is visited, the cost of the segment which is from the starting position to the current city is calculated and compared with the current best cost of the segment.

- If the cost of this segment is greater than the current best cost of the segment, this segment will be discarded.
- Otherwise, the next city in the segment is visited and the cost of the segment is calculated to this city. If the next city is the last of this segment, its cost is compared to the current best cost of the segment. Any better cost will be kept.

Sketch of applying *DP* to the tour is presented bellow.

```
Algorithm 2: ADPT(c)
Input: A individual T(c_1, c_2, ..., c_n)
Output: The optimization individual T
1.  Begin
2.    l ← length of the segment (sub-tour)
3.    k ← a random gen position
4.    S ← cites in the segment
5.    d_min ← cost of the initial segment
6.    SC ← Ø
7.    for each position p in the segment do
8.      for each random s in (S\SC) do
9.        c_p ← s
10.       ct ← cost of segment from position k-1 to p
11.       if ct > d_min then break end if
12.       add s to SC
13.       if (p is last position of the segment) then
14.         ct ← ct + cost of traveling between c_p and c_{k+1}
15.         if (ct < d_min) then
16.           d_min ← ct
17.           update T
18.           SC ← Ø
19.         end if
20.       end if
21.     end for
22.   end for
23.   return T
24. end
```

C. Computational results

1) Problem instances

The results are reported for the symmetric *TSP* by extracting benchmark instances from the *TSP*-lib [22]. The instances chosen for our experiments are eil51.tsp, kroA150.tsp, lin318.tsp, rat575.tsp, rat783.tsp, pr1002.tsp and nrw1379.tsp. The number of vertices: 51, 150, 318, 575, 783, 1002, 1379. Their weights are Euclidean distance in 2-D.

2) System setting

In the experiment, the system was run 10 times for each problem instance. All the programs were run on a machine with Intel Core 2 Duo T6400 2.0GHz, 2GB RAM, and were installed by C# language.

3) Experimental setup

To evaluation the proposed algorithm, we have two experiments. One is to re-implemented *GA* [5], the other is to implement the proposed algorithm, *CGADP*.

The parameters for two experiments:

Population size: $p_s = 100$

Mutation rate: $p_m = 1/n$

Crossover rate: $p_c = 0.9$

4) Experimental results

The experiments were implemented in order to compare GA [5] with *CGADP* in term of the min, mean, standard deviation values and running times.

Table 1 shows the results found by *GA* and *CGADP* algorithm for *TSP* instances (Eil51, Kro150, Lin318, Rat575, Rat738, Pr1002, Nrw1379). The experiment results show that the solutions found by *CGADP* algorithm are better than the ones found *GA* algorithm on the min, mean values and particularly on standard deviation values with large problem instances (Lin318, Rat738, Pr1002, Nrw1379). This proves that *CGADP* is quite effective with large instances.

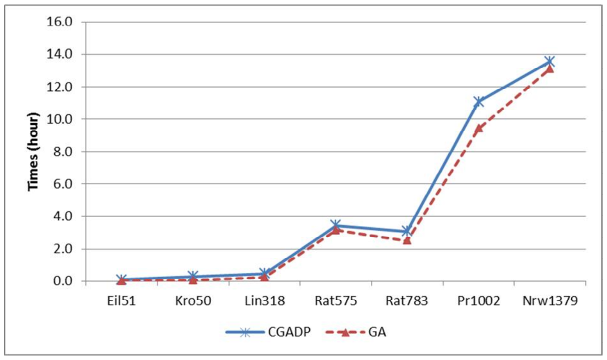


Figure 4. The average running time of the GA and CGADP algorithm on TSP-lib instances after 10 running times.

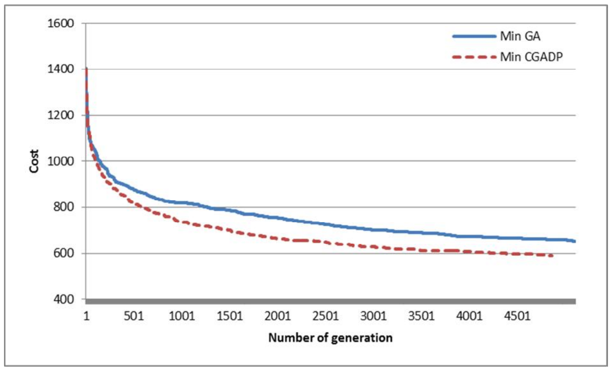


Figure 5. The convergence rate of GA and CGADP to find the best solution on the Eil51 instance

Figure 4 shows that *CGADP* takes more time than *GA* [5] to find the best solutions with the same number of fitness calculations.

Figures 5, 6 illustrate the convergence rates of *GA* and *CGADP* algorithms on *TSP*-lib instances. The diagrams show that *CGADP* converged faster than *GA*. This proves the effectiveness of *DP*-based local search.

In order to select the best value for the length of the segment which is used local search in *DP*, we experimented l = 2, 3, 4, 5, 6, 7. The figure 7 shows the dependence between the length of the segment, mean cost value found by 10 running times, and the running time of *CGADP* algorithm. According to the experiment in the figure 7, selecting (l = 5) is quite reasonable in our algorithm.

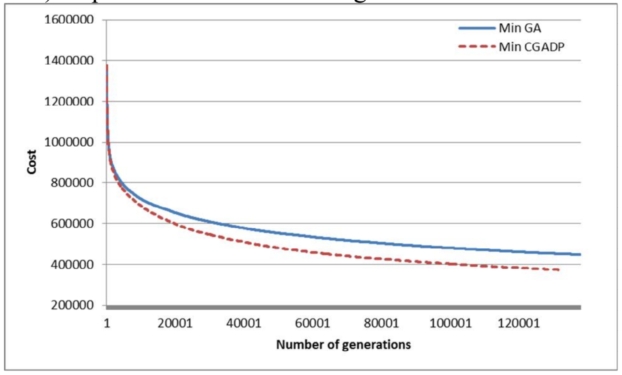


Figure 6. The convergence rate of GA and CGADP to find the best solution on the Nrw1379 instance.

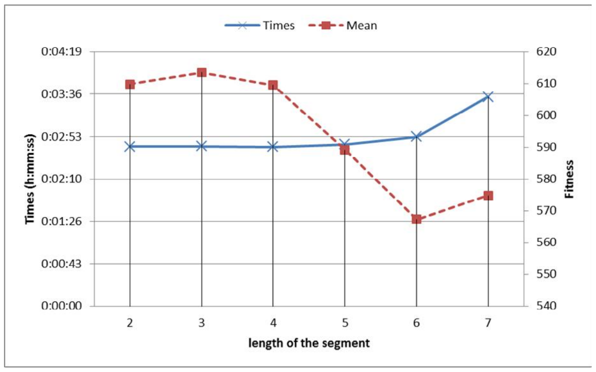


Figure 7. The relationship between the length of the segment, mean cost found by 10 running times, and the running time on Eil51 instance.

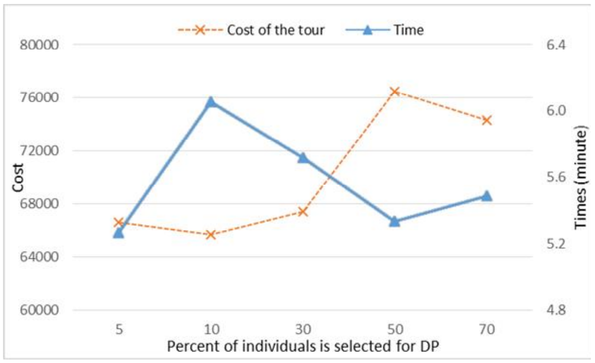


Figure 8. The relationship between the percent of individuals, mean cost found by 10 running times, and the running time on KroA150 instance.

In addition, we also tested the percentage of individuals to apply dynamic programming. We experimented with 5, 10, 30, 50 and 70 percent of the population size. The figure 8 shows that, *CGADP* algorithm is effective with 5 percent of the population size.

V. CONCLUSION

This paper gives a brief survey of combination between *GA* and *DP* for solving *TSP*. We discuss two ways of this combination: Using *DP* as a stage in *GA* and creating new genetic operators using *DP*. Two ways of this combination tend to attain better solution quality than *GA*.

Besides, we also implemented our combination of *DP* and *GA*. We experimented on 7 Euclidean instances derived from TSP-lib with the number of vertices: 51, 150, 318, 575, 783, 1002, 1379. On the *TSP*-lib instances, the best cost and the mean cost found by *CGADP* are better than the one found by *GA*. The results show that the combination algorithm approach can be attractive for solving *TSP*, particularly on large problem instances.

In the future, we are planning to improve the algorithm to reduce the running time. Moreover, a direction for further research could be a study of the Multi-objective *TSP*.

REFERENCES

[1] Held, M., Karp, R.M.: A dynamic programming approach to sequencing problems. Journal of the Society for Industrial and Applied Mathematics. vol. 10, pp. 196--210 (1962)

[2] Lin, S., Kernighan, B.W.: An effective heuristic algorithm for the traveling salesman problem. Operations Research, vol. 21, pp. 498--516 (1973)

[3] Kirkpatrick, S., Gelatt, C.D., Vechi, M.P.: Optimization by simulated annealing. Science, new series, vol. 220, pp. 671--680 (1983)

[4] Aarts, H., Lenstra, H.L, Lenstra, K.: Local Search in Combinatorial Optimization, pp. 215--310. Princeton University Press (1997)

[5] Eiben, A.E., Smith, J.E.: Introduction to Evolutionary Computing Natural Computing. Series 1st edition. Springer (2003)

[6] Haykin, S.: Neural Networks: A Comprehensive Foundation, 2nd Edition. Prentice-Hall (1999)

[7] Dorigo, M., Stutzle, T.: Ant Colony Optimization. Bradford Books, MIT Press (2004)

[8] Teodorovic, D., Lucic, P., Markovic, G., Dell'Orco, M.: Bee Colony Optimization: Principles and Applications. In: 8th Seminar on Neural Network Applications in Electrical Engineering, pp. 151--156. IEEE Press (2006)

[9] Adlema, L.M.: Molecular Computation of Solutions to Combinatorial Problems, vol. 266, pp. 1021—1024. Science (1994)

[10] Henry-Labordere, A.: The record balancing problem: A dynamic programming solution of a generalized traveling salesman problem. RAIRO Operations Research B2, 43--49 (1969)

[11] Fischetti, M., Salazar, J.J., Toth, P.: A branch-and-cut algorithm for the symmetric generalized traveling salesman problem. vol. 45, pp. 378--394. Operations Research (1997)

[12] Noon, C.E., Bean, J.C.: A Lagrangean based approach to the asymmetric generalized traveling salesman problem. Operations Research. 39, 623--632 (1991)

[13] Snyder, L.V., Daskin, M.S.: A random-key genetic algorithm for the generalized traveling salesman problem. European Journal of Operational research. 174, 38--53 (2006)

[14] Paquete, L., Stützle, T.: A Two-Phase Local Search for the Biobjective Traveling Salesman Problem. In: Second International Conference (EMO 2003), pp. 479--493. Springer, Heidelberg (2003)

[15] Chentsov, A.G., Korotayeva, A.G.: The dynamic programming method in the generalized traveling salesman problem. Mathematical and Computer Modeling. 25, 93--105 (1997)

[16] Yagiura, M., Ibaraki, T.: The Use of Dynamic Programming in Genetic Algorithms for Permutation Problems. European Journal of Operational Research. 92, 387--401 (1996)

[17] Lin, S., Kernighan, B.M.: An Effective Heuristic Algorithm for the Traveling-Salesman Problem. Operations Research. 21, 498--516 (1973)

[18] Nourolhoda Alemi Neissi, Masoud Mazloom: GLS Optimization Algorithm for Solving Travelling Salesman Problem. Second International Conference on Computer and Electrical Engineering (2009)

[19] Bernd Freisleben, Peter Merz: New Genetic Local Search Operators Traveling Salesman Problem. In: The 4th International Conference on Parallel Problem Solving from Nature, pp. 890--899. Springer, Heidelberg (1996)

[20] Bernd Freisleben, Peter Merz: New Genetic Genetic Local Search for the TSP: New Results. In: International Conference on Evolutionary Computation, pp. 159--164. IEEE Press (1997)

[21] Freisleben, B., Merz, P.: A Genetic Local Search Algorithm for Solving Symmetric and Asymmetric Traveling Salesman Problems. In: Proceedings of the 1996 IEEE International Conference on Evolutionary Computation, pp. 616—621 (1996)

[22] TSPLIB, http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/

[23] Renders, J.M.,Bersini, H.: Hybridizing genetic algorithms with hill-climbing methods for global optimization: two possible ways. In: Evolutionary Computation, 1994. IEEE World Congress on Computational Intelligence., Proceedings of the First IEEE Conference on. 1, 312 – 317 (1994)

[24] Wan-rong Jih, Hsu, J.Y.-J.: Dynamic vehicle routing using hybrid genetic algorithms. In: International Conference on Robotics & Automation. 1, 453 – 458 (1999).

TABLE 1. THE RESULTS FOUND BY GA AND CGADP ALGORITHM FOR TSP INTSTANCES.

No.	Problems	NCity	CGADP			GA		
			Min	Mean	Std	Min	Mean	Std
1	Eil51	51	502.3	589.2	40.5	612.7	652.6	36.0
2	Kro150	150	61076.8	66212.0	4310.3	72136.0	75507.8	2450.6
3	Lin318	318	150438.6	158795.5	5233.9	177488.0	185148.2	5696.8
4	Rat575	575	27896.5	29183.6	878.0	33191.4	34140.6	562.1
5	Rat783	783	44148.3	45241.3	640.8	53274.2	54335.8	797.9
6	Pr1002	1002	1627014.0	1682714.7	39166.9	1941224.0	2015872.1	39877.5
7	Nrw1379	1379	365986.2	373284.4	3868.7	434914.0	447344.2	6526.5

NCity: Number of city; Min: Minimum cost; Mean: Mean cost; Std: Standard Deviation of minimum cost