Int. J. Advance Soft Compu. Appl, Vol. 9, No. 2, July 2017
ISSN 2074-8523

Combination of Genetic Algorithm with Dynamic Programming for Solving TSP

Hemmak Allaoua

Computer science department,
University of Bejaia, Bejaia 06000,
AlgeriaComputer science department,
University of M'sila,M'sila 28130, Algeria
e-mail: hem_all@yahoo.fr

Abstract

This paper presents a combination of Genetic Algorithm (GA) with Dynamic Programming (DP) to solve the well-known Travelling Salesman Problem (TSP). In this work, DP is integrated as a GA operator with a certain probability. In specific, at a given GA generation, the individuals are subdivided into a number of equal segments of genes, and the shortest path on each segment is obtained by applying a DP algorithm. Since the computational complexity of the DP is O (k22k), it becomes of O(1) when k is small. Experimental analyses are conducted to investigate the impact and trade-offs among DP probability, segment size and time processing on the solution quality and computational effort. In addition, we will implement a basic GA approach to compare results and show the contribution of combination of combination approach. Experimental results on benchmark instances showed that the combined GA-DP algorithm reduces significantly the computational effort, produces a clearly improved solution quality and avoids early premature convergence of GA.

Keywords*: Combinatorial Optimization, Dynamic Programming, Evolutionary Computing, Genetic Algorithm, Traveling Salesman Problem.*

1 Introduction

It is clear that hybridizing methods to solve many cases of NP-Hard problems becomes a metaheuristics alone. Since the near aspect of metaheuristics, hybridizing exact method as DP with meta heuristic as GA give certainly good results that hybridizing just meta heuristics between each other, despite the time cost inducted by using exponential complexity method as DP. In the other hand, the

use of a model of NP-Hard problem such as TSP still necessary to show the efficiency of the proposed approach. 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 combination Genetic Algorithm (GA) with Dynamic Programming (DP) for solving TSP on 10 Euclidean instances derived from TSP-lib. Experimental results are reported to show the efficiency of the experimented algorithm comparing to, on one hand to basic GA results, and, in other hand, to the existing results. In this paper, we present a combination of GA with DP (called GA-DP) for solving this problem. In GA-DP, DP is used as genetic operator as crossover and mutation. Therefore, in each generation, according a specific probability $P_{DP,}$ each solution is subdivided on equal segments of size k, (we will take k=10; 15; 20 and P_{DP} = 0.5, 0.6, 0.7, 0.8). For each segment, DP is applied to compute the shortest path for this segment. We experiment this approach on 10 Euclidean instances derived from TSP-lib [22] and compare the result with basic GA. Because of the stochastic aspect of GA, for each TSP instance, both GA-DP and basic GA are applied for five times and average values are reported on result tables.

Our motivation emanate from that traveling salesman problem is an important problem in computing fields and has many applications in the real-world such as scheduling, vehicle routing, economic models, 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.

The next section introduces the literature of the hybridizing Genetic Algorithms with Dynamic Programming for Solving TSP. In section III, we propose basic GA approach. Section VI presents the combination GA-DP approach for solving TSP. The details of our experiments and the computational and comparative results are given in section V. The paper concludes with section VI with some discussions on the future perspectives of this work.

2 Literature

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:

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

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 usually 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, researchers have concerned approximation approaches in recent years. Many approximation approaches were proposed for solving TSP such as 2- opt, 3-opt [2], simulated annealing [3], tabu search [4,28]; 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 [26,27].

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.

3 Basic GA presentation

Genetic algorithms, introduced by J. Holland (1975), are inspired from the Darwin evolution theory: in the population evolution, the best individuals, which are more adapted to their environment, can outlive for a long time, on the on other hand, the individuals, which are not fits to their environment, disappear with the passage of

generations. Therefore, its chromosome and an appropriate fitness function to be defined to evaluate individuals code each individual. Firstly, GA consists to randomly generate initial population, then, genetic operators (selection, crossover, mutation), within specified probabilities, are applied to produce a new generation that considered best than its previous. This process must be iterative for a great number of generations as shown as follow:

Begin

Initialization;

Evaluation;

Repeat

Selection;

Crossover;

Mutation;

Evaluation;

Until (Criteria Stopping);

End.

However, the individuals encoding, fitness function, selection method, probability crossover, probability mutation and criteria stopping depend of the treated problem. These are the GA parameters. So they must be carefully chosen, because they can considerably affect the solution quality and the rate of GA convergence. Good choice of these parameters often avoids premature convergence of GA.

4 Proposed approach GA-DP

As we have reported in the literature above, there are many ways for hybridizing GA with DP. In our case, we integrate DP as a GA operator with specific probability P_{DP}. Exactly like crossover and mutation operators. This new phase will be added just before evaluation phase as follow:

Begin

Initialization;

Evaluation;

Repeat

Selection;

Crossover;

Mutation;

Apply_DP;

Evaluation;

Until (Criteria Stopping);

End.

However, the essential of our work consists to subdivide the individual on equal segments sized k (k $\in$ {5,10,15,20}) and applying DP to compute the shortest path of each segment. General algorithm of this step will be as follow:

Algorithm Apply_DP (integer i; P_{DP})

Begin

Generate a probability P_i;

If ($P_i \leq P_{DP}$) For (each segment S of individual i) Compute_shortest_path(S);

End.

The DP algorithm (C*ompute_shortest_path(S)*) consists to run through the current generation. For each individual i, we generate a probability P_i. If $P_i \leq P_{DP}$, the following process is applied.

For a subset of k cities (k=5,10, 15 or 20) S $\subseteq$ {1,2,...,n}, S= {i_1,,...,i_k} that includes i_1, and j$\in$S, let C(S,j) be the length of the shortest path visiting each node in S exactly once, starting at i_1 and ending at j.

When |S| > 1, we define C(S, 1) = ∞ since the path cannot both start and end at i_1. Now, let's express C(S,j) in terms of smaller sub-problems (Bellman principle). We need to start at i_1 and end at j; what should we pick as the second-to-last city? It has to be some i $\in$ S, so the overall path length is the distance from i_1 to i, namely, C(S−{j}, i), plus the length of the final edge, $d_{i,j}$. We must pick the best such i:

C(S,J) = $\min_{i \in S-\{J\}} C(S-\{j\}, i) + d_{i;j}$ (Bellman principle)

The sub-problems are ordered by |S|. Here is the code :

$C(\{i_1\},i_1) = 0$
for m = 2 to k:
  for all subsets $S \subseteq \{1,2,...,k\}$ of size m and containing i_1:
    $C(S,i_1) = \infty$
    for all $j \in S, j \neq i_1$: $C(S,j) = \min_{i \in S-\{J\}} C(S-\{j\}, i) + d_{i;j} : i \in S, i \neq j\}$
return permutation(S) having $\min_i C(\{i_1, \dots, k\}, j\})$

There are at most $k.2^k$ sub-problems, and each one takes linear time to solve. The total running time is therefore $O(k^2 2^k)$. .

5 Computational results and comments

5.1 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, eil76.tsp, eil101, kroA100, kroA150, kroA200, lin318, rat575, rat783, pr1002. Their weights are Euclidean distance in 2-D.

5.2 System configuration

In the experiment, the system was run 10 times for each problem instance. All the programs were run on a machine with Intel i3 2.3 GHz, 4 GB RAM, and were installed by C# language.

5.3 Implementation setting

In order to evaluate GA-DP approach, compare it with basic GA and use TSP-Lib, according the well-known GA parameters values and our own results obtained for other problems, we have opted for the following configuration:

Input:

- Let n the size of the TSP instance, the number of cities. According the instances chosen below, according to chosen TSP instances, we have : $n \in \{51,76,100, 101, 150, 200 , 318 , 575 , 783 , 1002\}$.

Parameters:

- Population Initialization in GA: by uniform law.
- Population size (number of individuals in each generation) = 10000 log n.
- Number of generations (iterations in GA) =50000 log n. (criteria stopping).
- Selection operator in GA: by fortune wheel method.
- Crossover probability, p_c=0.8.
- Mutation probability, p_m=0.03.
- Fitness function(i) = 1-Obj(i)/ΣObj(i)

Variables:

- Size of segment in GA-DP, $k \in \{5,10,15, 20\}$.
- DP probability P_{DP}, $P_{DP} \in \{ 0.5 , 0.6 , 0.7, 0.8\}$.

Output:

- Opt.= optimal solution provided by TSP-LIB.
- Sol.= near solution obtained by our approaches (GA or GA-DP).
- T CPU = average of times processing in second for 10 runs of each TSP instance.
- *Ratio* of *Performance* to *Deviation* = RPD = $\{(\text{Sol} - \text{Opt})/\text{Opt}\} * 100$.

5.4 Experimental results

In the first step, we show in table 1 below, sample example of using DP to find shortest path of set S with smaller size k:

Table 1: Processing time of DP algorithm with smaller size.

k	5	10	15	20
CPU time (s)	0.12	0.020	0.641	21.748

This table shows clearly that DP is an exponential algorithm but it is O(1) for smaller size. When used in GA as operator that will add in worst case:

*Number of generations * number of genes*$k^2 2^k = k^2 2^k (n^2 \log^2 n)$ elementary operations.*

This proves that when integrating DP in GA as operator, the hybrid approach obtained still polynomial as GA despite of its cost in time processing.

In the second step, we show in table 2, the results obtained by basic GA using parameters described above;

Table 2: Results of basic GA.

Instance	Opt.	Sol.	RPD	T. cpu
eil51	426	451	5.87	40.54
eil76	538	574	6.69	66.52
eil101	629	660	4.93	94.02
kroA100	21282	36345	70.78	88.58
kroA150	26524	42615	60.67	140.8
kroA200	29368	51541	75.50	179.05
lin318	42029	73202	74.17	345.29
rat575	6773	20798	207.07	944.6
rat783	8806	24862	182.33	1190.3
pr1002	259045	856431	230.61	1446

This table shows clearly that, when n is bigger, RDP becomes larger. These results will be later compared to both the best and worst cases of hybrid approach GA-DP.

The third step consists to look for the impact of k (segment size) on solution quality and time processing in GA-DP approach (Tables 3 to 5):

Table 3: Results of GA-DP approach for k=5.

		P_{DP}=0.6			P_{DP}=0.7			P_{DP}=0.8		
instance	Opt.	Sol.	RPD	T. cpu	Sol.	RPD	T. cpu	Sol.	RPD	T. cpu
eil51	426	449	5.40	53.55	449	5.40	54.33	442	3.76	55.11
eil76	538	569	42.94	95.40	568	5.58	97.13	544	1.12	98.87
eil101	629	654	3.97	145.03	652	3.66	148.09	641	1.91	151.15
kroA100	21282	34201	60.70	138.58	33547	57.63	141.58	33129	55.67	144.58
kroA150	26524	39650	49.49	253.30	39186	47.74	260.05	36775	38.65	266.80
kroA200	29368	49021	66.92	379.05	48857	66.36	391.05	47360	61.26	403.05
lin318	42029	57005	35.63	850.91	52491	24.89	881.25	53087	26.31	911.58
rat575	6773	15128	123.36	2597.73	15005	121.54	2696.91	15018	121.73	2796.10
rat783	8806	14200	61.25	4255.75	14100	60.12	4439.67	13825	57.00	4623.60
pr1002	259045	589681	127.64	6466.02	541806	109.16	6767.22	487365	88.14	7068.42

Table 4: Results of GA-DP approach for k=10.

		P_{DP}=0.6			P_{DP}=0.7			P_{DP}=0.8		
instance	Opt.	Sol.	RPD	T. cpu	Sol.	RPD	T. cpu	Sol.	RPD	T. cpu
eil51	426	443	3.99	79.56	441	3.52	81.12	438	2.82	81.90
eil76	538	551	2.42	153.16	552	2.60	156.63	544	1.12	158.36
eil101	629	645	2.54	247.04	645	2.54	253.16	641	1.91	256.22
kroA100	21282	34167	60.54	238.58	33594	57.85	244.58	32518	52.80	247.58
kroA150	26524	37508	41.41	478.30	36843	38.90	491.80	36408	37.26	498.55
kroA200	29368	47891	63.07	779.05	47210	60.75	803.05	46337	57.78	815.05
lin318	42029	55843	32.87	1862.15	53008	26.12	1922.82	52994	26.09	1953.16
rat575	6773	14120	108.47	5903.98	13975	106.33	6102.35	13412	98.02	6201.54
rat783	8806	13914	58.01	10386.64	14250	61.82	10754.49	12807	45.43	10938.42
pr1002	259045	550890	112.66	16506.06	528045	103.84	17108.46	467290	80.39	17409.66

Table 5: Results of GA-DP approach for k=20.

		PDP=0.6			PDP=0.7			PDP=0.8		
instance	Opt.	Sol.	RPD	T. cpu	Sol.	RPD	T. cpu	Sol.	RPD	T. cpu
eil51	426	439	3.05	92.56	436	2.35	94.12	433	1.64	94.90
eil76	538	551	2.42	182.04	548	1.86	185.51	548	1.86	187.24
eil101	629	645	2.54	298.04	646	2.70	304.16	639	1.59	307.22
kroA100	21282	32837	54.29	288.58	30856	44.99	294.58	30890	45.15	297.58
kroA150	26524	38652	45.72	590.80	34278	29.23	604.30	33081	24.72	611.05
kroA200	29368	42673	45.30	979.05	41941	42.81	1003.05	41150	40.12	1015.05
lin318	42029	50558	20.29	2367.77	48792	16.09	2428.44	45552	8.38	2458.78
rat575	6773	12590	85.89	7557.10	11780	73.93	7755.48	11499	69.78	7854.66
rat783	8806	13666	55.19	13452.08	13251	50.48	13819.93	12074	37.11	14003.86
pr1002	259045	511211	97.34	21526.08	499802	92.94	22128.48	451007	74.10	22429.68

Results reported in these tables show that GA-DP becomes more efficient for biggest TSP instances size. On the other hand, when k increase, near solution is better but processing time becomes more important. Also, the same remark when P_{DP} increase. The next step consists to show what is the best configuration in term

of two components (k,P_{DP}). To find the best combination of these parameters, we will choose three instances from the above ten instances used in our work, and then we build the curves that represent solution and time processing in function of the couple (k,P_{DP}) (fig. 1 to 6).

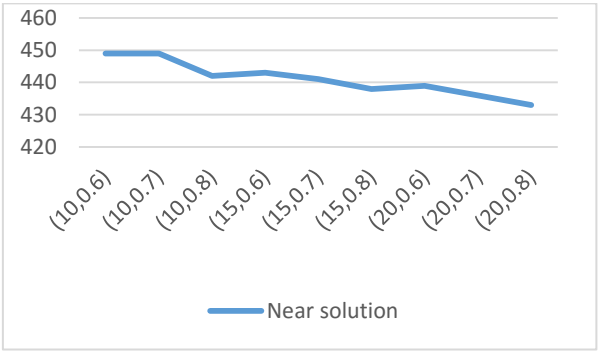

Fig.1: Representation of near solution depending on (k, P_{DP}) for eil51.

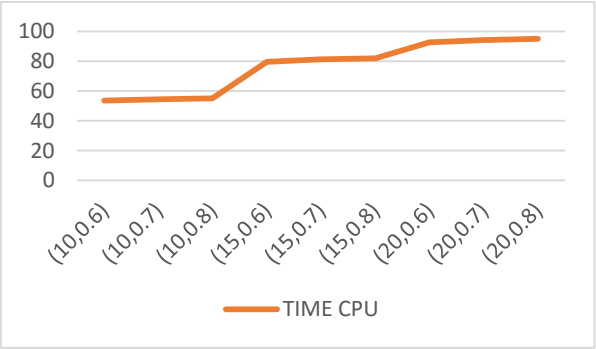

Fig.2: Representation of processing time depending on (k, P_{DP}) for eil51.

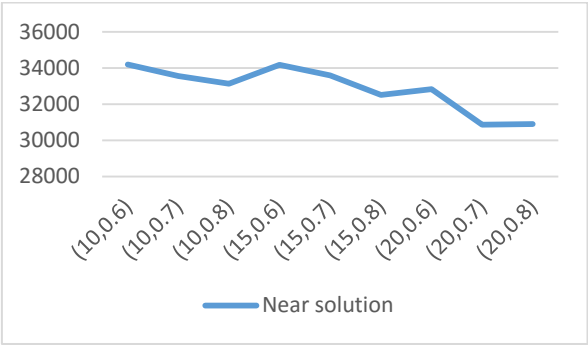

Fig.3: Representation of near solution depending on (k, P_{DP}) for kroA100.

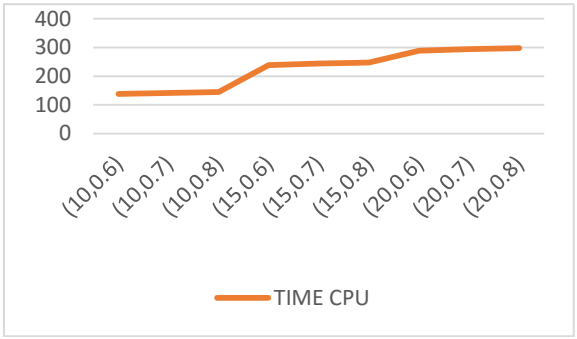

Fig.4: Representation of processing time depending on (k, P_{DP}) for kroA100.

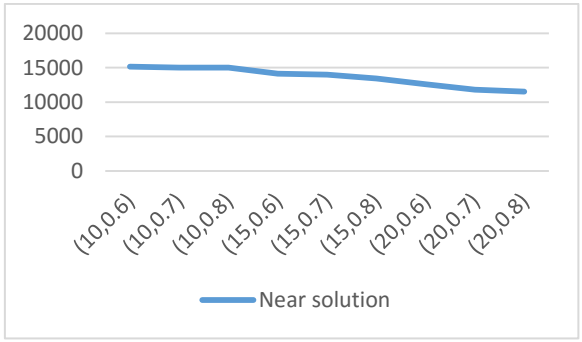

Fig.5: Representation of near solution depending on (k, P_{DP}) for rat575.

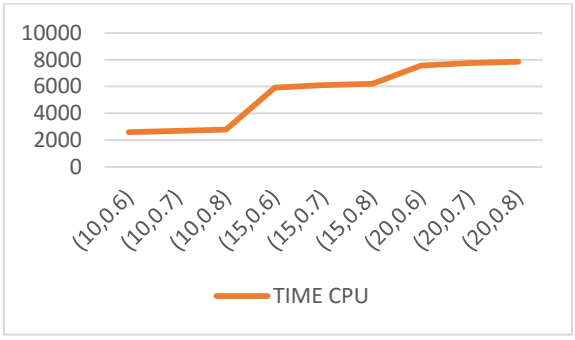

Fig.6: Representation of near solution depending on (k, P_{DP}) for rat575.

The shape of these curves show that when both k and P_{DP} increase, near solution becomes better but time processing becomes longer.

To more interpret these curves, we consider the portions where both near solution and time processing are lower. For these 3 instances, this portion is reported in the set of couples {(15,0.7); (15,0.8); (20,0.6); (20,0.7)}. This set represent a

compromise of segment size and DP probability that realize acceptable results in terms of solution quality and time processing.

In the last step of our comparative study, we will dress the curves (figure 7) of near solutions of both basic GA and case of GA-DP depending on 10 TSP instances treated in this work. That to compare GA-DP with DP approaches efficiency.

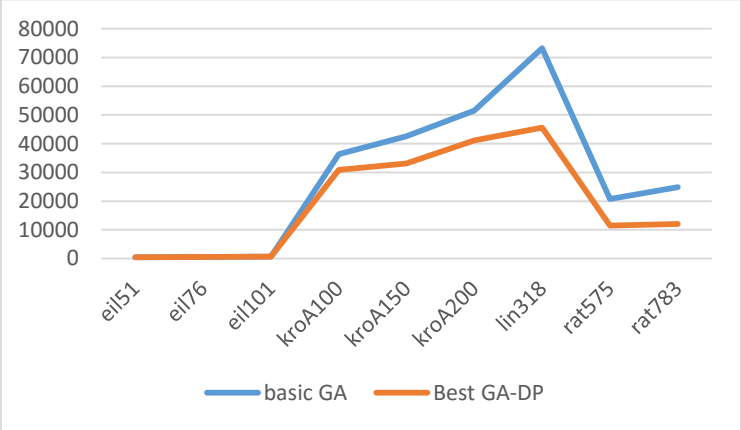

Fig.7: Representation of near solution depending on TSP instances for basic GA and best GA-DP.

This curve shows that for instances with biggest size, the gap between GA and GA-DP becomes more significant, certainly that has a significant cost in term of time but it still polynomial. For smaller instances with smaller size, this gap becomes weaker.

6 Conclusion and perspectives

In this work, we have treated a variant of hybridizing genetic algorithm with dynamic programming. We have chosen TSP since its significance in optimization field in many situations where it can be used as a model to solve NP-Hard optimization problems. In our approach, DP is integrated as a phase in the genetic algorithm process with studied probability called P_{DP} exactly like other genetic operators such as selection, crossover, and mutation. Therefore, when this phase is applied on an assumed generation, each chromosome is subdivided on a great number of segments of size k, which was studied here. Since the exponential complexity of DP, only small sizes of k are tested, that keep the hybrid approach in polynomial class. Each segment is ordered by DP algorithm just before evaluation phase of GA. According the results shown above we note that:

- Integrating DP improve considerably the quality solution, in contrast it has an impact on the processing time compared to basic GA.
- The adjustment of both size of segment k and DP probability can clearly converge to good compromise between solution quality and time processing.

- In general, cases, hybridizing exact method with metaheuristic give certainly best results than hybridizing only metaheuristics.

In other hand, according the cases treated here and the tests achieved, we propose the following perspectives:

- The proposed approach can be used on other NP-Hard problems of operational research as scheduling, linear programming, transport problems; it could give good results better than existing metaheuristics.
- The fast evolution of computer power will allow treating biggest sizes of segment that could open new horizons of this kind of hybridization.
- The introduction of scaling and sharing operations in GA will clearly improve the GA efficiency despite their impact against time processing o GA.

Finally, we conclude that, by the present work, we have tried to provide a modest contribution in the field of hybridization of exact method with meta heuristic as GA and DP to seek for the best configuration that give better than using only meta heuristic. We consider that the technology evolution will favorite hybridizing approach as alternative to metaheuristics in optimization problems. We aimed to provide a new tool at the disposal of researchers in the field that could participate to improve their work results.

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).
[25] PHAM D. T. et al. : A Survey on Hybridizing Genetic Algorithm with

Dynamic Programming for Solving the Traveling Salesman Problem. In: 2013 Online Proceedings on Trends in Innovative Computing (ICT).

[26] Premalatha, K. and Natarajan, A.M. Hybrid PSO and GA Models for Document Clustering. International Journal of Advances in Soft Computing and its Application, 2,3 (2010), 1-20.

[27] Bahesti, Z. and Shamsuddin, S.M. A Review of Population Based Meta-Heuristic Algorithm. International Journal of Advances in Soft Computing and its Application, 5, 1(2013), 1-35.

[28] Hemmak Allaoua, Bouderah Brahim, Sieve Algorithm - A New Method for Optimization Problems. International Journal of Advances in Soft Computing and its Application, 5, 2(2013), 1-15.