20

Heuristic Algorithms for Solving Bounded Diameter Minimum Spanning Tree Problem and Its Application to Genetic Algorithm Development

Nguyen Duc Nghia and Huynh Thi Thanh Binh
Ha Noi University of Technology
Viet Nam

1. Introduction

The bounded diameter minimum spanning tree (*BDMST*) problem is a combinatorial optimization problem that appears in many applications such as wire-based communication network design when certain aspects of quality of service have to be considered, in ad-hoc wireless network (K. Bala, K. Petropoulos, T.E. Sterm, 1993) and in the areas of data compression and distributed mutual exclusion algorithms (K. Raymond, 1989; A. Bookstein, S. T. Klein, 1996). A more comprehensive discussion of the real-world applications of BDMST was given in Abdalla's seminal dissertation (Abdalla, 2001).

Before the *BDMST* problem can be formally stated, we need some definitions relating to tree diameter and center. Given a tree *T*, the maximal eccentricity of vertex v is the length (measured in the number of edges) of the longest path from v to other vertices. The diameter of a tree *T*, denoted as diam(*T*), is the maximal eccentricity over all nodes in *T* (i.e the length of maximal path between two arbitrary vertices in *T*). Suppose that a diameter of a tree is defined by the path v_0, v_1, v_2,…, $v_{[k/2]}$, $v_{[k/2]+1}$, …, v_k. If k is even then $v_{[k/2]}$ is called a center of the tree. If k is odd then $v_{[k/2]}$ and $v_{[k/2]+1}$ are centers of the tree. In that case, the edge ($v_{[k/2]}$, $v_{[k/2]+1}$) is called a center edge.

Let $G = (V, E)$ be a connected undirected graph with positive edge weights $w(e)$. The *BDMST* problem can be formulated as follows: among spanning trees of *G* whose diameters do not exceed a given upper bound $k \geq 2$, find the spanning tree with the minimal cost (sum of the weights on edges of the trees). As in almost all studies of the *BDMST* problem, and without lost of generality, we will assume that *G* is a complete graph.

More precisely, the problem can be stated as:

Find a spanning tree *T* of *G* that minimizes

$$W(T) = \sum_{e \in T} w(e)$$

subject to

$$\text{diam}(T) \leq k .$$

Source: Advances in Greedy Algorithms, Book edited by: Witold Bednorz,
ISBN 978-953-7619-27-5, pp. 586, November 2008, I-Tech, Vienna, Austria

This problem is known to be *NP*-hard for $4 \le k < |V|-1$ (M.R.Garey & D.S.Johnson, 1979).
In this chapter, we introduce the heuristic algorithms for solving *BDMST*: *OTTC* (Abdall, 2001), *RGH* (R.Raidl & B.A.Julstrom, 2003), *RGH*$_1$ (Binh et.at, 2008a), *RGH-I* (A. Singh & A.K. Gupta, 2007), *CBRC* (Binh et al., 2008b). In order to inlustrate the effectiveness of the proposed algorithms, we apply them for initializing the population of our new genetic algorithm with multi-parent recombination operator for solving given problem. Then results of computational experiments are reported to show the efficiency of proposed algorithms.
The chapter is organized as follows. In the next section (section 2), we briefly overview works done in solving *BDMST* problems. Section 3 deals with new heuristic algorithm for solving *BDMST* problem. Section 4 describes our new genetic algorithm which uses heuristic algorithms that already presented in previous section to solve *BDMST* problem. The details of experiments and the comparative computational results are given and discussed in the last section of the chapter.

2. Previous work on the BDMST problem

Techniques for solving the *BDMST* problem may be classified into two categories: exact methods and inexact (heuristic) methods. Exact approaches for solving the *BDMST* problem are based on mixed linear integer programming (N.R.Achuthan et al., 1994), (L Gouveia et al., 2004). More recently, Gruber and Raidl suggested a branch and cut algorithm based on compact 0-1 integer linear programming (M. Gruber & G.R. Raidl, 2005). However, being deterministic and exhaustive in nature, these approaches could only be used to solve small problem instances (e.g. complete graphs with less than 100 nodes).
(Abdalla et al., 2000) presented a greedy heuristic algorithm - the One Time Tree Construction (*OTTC*) for solving the *BDMST* problem. *OTTC* is based on Prim's algorithm in (R. Prim, 1957). It starts with a set of vertices, initially containing a randomly chosen vertex. The set is then repeatedly extended by adding a new vertex that is nearest (in cost) to the set, as long as the inclusion of the new node does not violate the constraint on the diameter of the tree. This algorithm is time consuming, and its performance is strongly dependent on the starting vertex.
Raidl and Julstrom proposed in (G.R. Raidl & B.A. Julstrom, 2003) a modified version of *OTTC*, called Randomized Greedy Heuristics (*RGH*). *RGH* starts from a centre by randomly selecting a vertex and keeping it as the fixed center during the search. It then repeatedly extends the spanning tree from the center by adding a randomly chosen vertex from the remaining vertices, and connecting it to a vertex that is already in the tree via an edge with the smallest weight. The obtained results showed that on Euclidean instances *RGH* performs better than *OTTC*, whereas on non-Euclidean instances the situation is reversed.
RGH could be summarized in the following pseudo-code (G.R. Raidl & B.A. Julstrom, 2003)

```
T  ← ∅;
U  ← V;
v0  ← random(U);
U ← V – {v0};
C ← {v0};
depth[v0] ← 0;
if (odd(k)) {
       v1  ← random(U);
```

```
            T  ← {(v0,v1)};
            U ← U – {v1};
            C ← C ∪ {v1};
            depth[v1] ← 0;
}
while (U ≠ ∅) {
            v ← random(U);
            u ← argmin {c(x,v): x ∈ C};
            T ← T ∪ {(u,v)};
            U ← U – {v} ;
            depth[v] ← depth[u] + 1;
            if (depth[v] < [k/2])
                        C ← C ∪ {v} ;
}
return T;
```

Raidl and Julstrom proposed a genetic algorithm for solving *BDMS*T problems which used edge-set coded (G.R. Raidl & B.A. Julstrom, 2003) (*JR-ESEA*) and permutation-coded representations for individuals (B.A. Julstrom & G.R. Raidl, 2003) (*JR-PEA*). Permutation-coded evolutionary algorithms were reported to give better results than edge-set coded, but usually are much more time consuming. Another genetic algorithm, based on a random key representation, was derived in (B.A. Julstrom, 2004), sharing many similarities with the permutation-coded evolutionary algorithms. In (M. Gruber & G.R. Raidl, 2005), Gruber used four neighbourhood types to implement variable neighbourhood local search for solving the *BDMST* problem. They are: arc exchange neighbourhood, level change neighbourhood, node swap neighbourhood, and center change level neighbourhood. Later, (M. Gruber et al., 2006), re-used variable neighbourhood searches as in (M. Gruber & G.R. Raidl, 2005), embedding them in Ant Colony Optimization (*ACO*) and genetic algorithms for solving the *BDMST* problem. Both of their proposed algorithms (*ACO* and *GA*) exploited the neighbourhood structure to conduct local search, to improve candidate solutions. In (Nghia & Binh, 2007), Nghia and Binh proposed a new recombination operator which uses multiple parents to do the recombination in their genetic algorithm. Their proposed crossover operator helped to improve the minimum and mean weights of the evolved spanning trees.
More recently, in (A. Singh & A.K. Gupta, 2007), Alok and Gupta derived two improvements for *RGH* heuristics (given in (G.R. Raidl & B.A. Julstrom, 2003)) and some new genetic algorithms for solving *BDMST* problems (notably the *GA* known as *PEA-I*). *RGH-I* in (A. Singh & A.K. Gupta, 2007) iteratively improves the solution found with *RGH* by using level change mutation. It was shown in (A. Singh & A.K. Gupta, 2007) that *RGH-I* has better results than all previously-known heuristics for solving the *BDMST* problem. *PEA-I* employs a permutation-coded representation for individuals. It uses uniform order-based crossover and swap mutation as its genetic operators. *PEA-I* was shown to be the best *GA* of all those tried on the *BDMST* problem instances used in (A. Singh & A.K. Gupta, 2007). In (Binh et al., 2008a), Binh et al., also implement another variant of *RGH*, which is called *RGH*$_1$. *RGH*$_1$ is similar to *RGH*, except that when a new vertex is added to the expanding spanning tree, it is chosen at random, and connected to a randomly chosen vertex that is already in the spanning tree.

3. New greedy heuristic algorithm (center-based recursive clustering)

Our new greedy heuristics is based on *RGH* in (G.R. Raidl & B.A. Julstrom, 2003) and *NRGH* in (Nghia and Binh, 2007), called *CBRC*. We extend the concept of center to every level of the partially constructed spanning tree. The algorithm can be seen as recursively clustering the vertices of the graph, in that every in-node of the spanning tree is the center of the sub-graph composed of nodes in the subtree rooted at this node. It is inspired from our observation (and other such as in (A. Abdalla et.al, 2000), (G.R. Raidl and B.A. Julstrom, 2003) that good solutions to the *BDMST* problem usually have "star-like structures" as can be seen (for a Euclidean graph) in Figure 1.

In a star-like structure, the vertices of the graph are grouped in clusters, and the clusters are connected by a link between their centers. Pseudocode for the new heuristic based on this observation, known as Center-Based Recursive Clustering (*CBRC*), is presented below:

```
   1. $T \leftarrow \varnothing$;
   $v_0 \leftarrow$ Choose_a_Center($V$)
   $U \leftarrow V - \{v_0\}$;
   $C \leftarrow \{v_0\}$;
   depth[$v_0$] $\leftarrow 0$;
  If $k$ is odd then
   {
    $v_1 \leftarrow$ Choose_a_Center($U$)
        $T \leftarrow \{(v_0, v_1)\}$;
        $U \leftarrow U - \{v_1\}$;
        $C \leftarrow C \cup \{v_1\}$;
        depth[$v_1$] $\leftarrow 0$;
    }
 2. //Group vertices in U into cluster(s)
    //with centers at $v_0$ or $v_1$
     For each node $w$ in $U$ do
     {
       If $k$ is even then
       {
        $w$ becomes child of $v_0$ ;
        depth[$w$]=1;
        $T \leftarrow T \cup \{(w,v_0)\}$;
        }
       Else // $k$ is odd
        If Distance($w,v_0$) $\le$ Distance($w,v_1$) then
          {
            $w$ becomes child of $v_0$;
            depth[$w$]=1;
            $T \leftarrow T \cup \{(w,v_0)\}$;
          }
        Else
          {
            $w$ becomes child of $v_1$;
            depth[$w$]=1;
            $T \leftarrow T \cup \{(w,v_1)\}$;
          }
      } //end for
  3. Loop
```

```
        V= set of leaves in U with depths < ⌊k/2⌋;
        v= Choose_a_Center(V);
        if(v is empty)
          Break; // Jump out of the loop
        U = U − {v};
    For each leaf node w in U do
     {
      If Distance(w,v) ≤ Distance(w, parent(w)) then
         w becomes child of v;
         depth[w]=depth[v] +1;
         T=T -{(w,parent(w))}+{(w,v)};

     }
```

The algorithm above is a general framework for *CBRC*. It employs two abstract functions, namely, *Choose_a_Center* and *Distance*. The implementations of these functions are expected to affect the performance of the heuristics, and the best choice could depend on the problem instance. We propose below some possible implementations of these two functions.

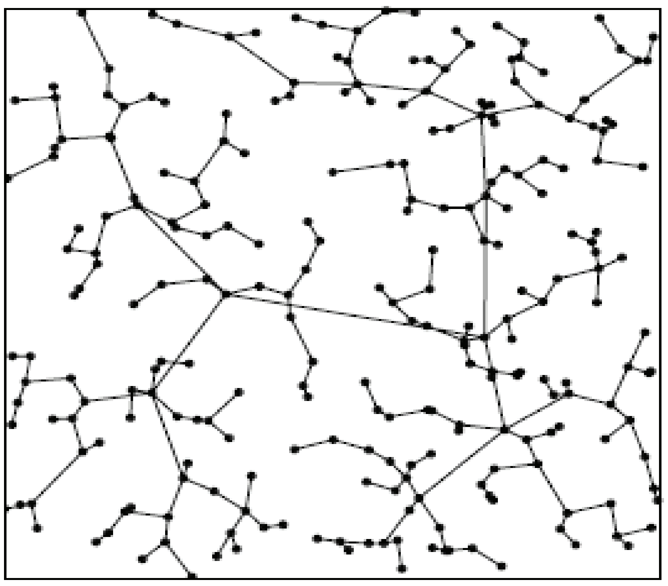

Fig. 1. A "star-like" structure of a typical solution to the BDMST problem.

Implementations of *Choose_a_Center* function:

- v is a center of U if $\sum w \in U$ Distance(v, w) $\rightarrow$ min. If there is more than one such v then choose from them randomly.
- Rank all vertices in U according to $\sum w \in U$ Distance(v, w), then choose v randomly from the first h% of the vertices.
- Conduct h-tournament selection, $\sum w \in U$ Distance(v, w) as the vertex for v.
- Choose v randomly (i.e. it does not depend on Distance at all).

Implementations of the Distance function:

- Distance(u, v) = $c(u, v)$.
- Distance(u, v) = cost the of shortest path between u and v (used for Non-Eclidean graphs).

It can be seen from the pseudo-code of *CBRC* that none of the combinations of *Distance* and *Choose_a_Center* from the above implementations increase the asymptotic computational complexity of the heuristic to more than $O(n^3)$. It is also possible to apply postimprovement, as proposed in (A. Singh and A.K. Gupta, 2007) to *CBRC* just as for *RGH*. The resulting heuristic is known as *CBRC-I*. In the next section, *CBRC* is tested on some benchmark Euclidean instances of the *BDMST* problem.

4. Proposed genetic algorithm

Genetic algorithm has proven effective on *NP*-hard problem. Much works research on NP-hard problem, particularly in problems relating to tree have been done. Several studies proposed representations for tree (J.Gottlieb et al., 2000), (G.R.Raidl & B.A.Julstrom, 2003), (B.A.Julstrom & G.R.Raild, 2003), (B.A.Julstrom, 2004), (Martin Gruber et al., 2006), (Franz Rothlauf, 2006). This section presents the genetic algorithm for solving *BDMST* problem.

4.1 Initialization

Use *OTTC*, RGH_1, *CBRC, RGH* heuristic algorithms described above for initializing population and edge list for chromosome code.

4.2 Recombination operator

Using *k*-recombination operator as in (Nghia and Binh, 2007).

4.3 Mutation operator

Using four mutations operators: edge delete mutation, center move mutation, greedy edge replace mutation, subtree optimize mutation as in (G.R.Raidl & B.A.Julstrom, 2003).

5. Computational results

5.1 Problem instances

The problem instances used in our experiments are the *BDMST* benchmark problem instances used in (G.R. Raidl & B.A. Julstrom, 2003), (A. Singh & A.K. Gupta, 2007), (Nghia & Binh, 2007), (Binh et al., 2008a) . They are Euclidean instances. All can be downloaded from http://www.sc.snu.ac.kr/~xuan/BDMST.zip. Euclidean instances are complete random graphs in the unit square. We chose the first five instances of each problem size on Euclide instances (number of vertices) n = 100, 250, 500, and 1000, the bounds for diameters being 10, 15, 20, 25 correspondingly (making up 20 problem instances in total).

5.2 Experiment setup

We created two sets of experiments. In the first set of experiment, we compare the performance of the heuristic algorithms: *OTTC*, *RGH*, RGH_1, *CBRC*. The detail of the comparison between other heuristic algorithm for solving *BDMST* problem such as *CBTC*, *RGH-I*, *CBRC-I* can be refered to (Binh et al., 2008b), (A. Singh and A.K. Gupta, 2007).
There are several heuristic algorithms for solving *BDMST* problem as mentioned above but no research has concerned with their effectiveness in application to develop hybrid genetic algorithm. Therefore, in second set of experiment, we will try to fix this problem.
In the second set of experiment, we tested six genetic algorithm algorithms for solving *BDMST* problem. All of the genetic algorithms use recombination and mutation operator mentioned in section 4 but initialized by different heuristic algorithm. GA_1, GA_2, GA_3 uses

CBRC, OTTC, RGH$_1$ algorithm correspondent for initializing the population. *GA*$_4$ uses *CBRC, OTTC, RGH*$_1$ *, RGH* for initialization the population with the same rate for each heuristic. *GA*$_5$ uses *RGH*$_1$, *CBRC* for initializing, the rate of them in the population are 30 and 70. *GA*$_6$ uses *RGH*$_1$, *OTTC*, *CBRC* for initializing, the rate of them in the population are 35, 35 and 30.

	GA_1	GA_2	GA_3	GA_4	GA_5	GA_6
CBRC	100%	0%	0%	25%	70%	30%
RGH	0%	0%	0%	25%	0%	0%
OTTC	0%	100%	0%	25%	0%	35%
RGH_1	0%	0%	100%	25%	30%	35%

Fig. 2. The rate of the heuristic algorithms use for initialization of the population in each experiment genetic algorithm

5.3 System setting

In the first experiment, the system was run 300 times for each instances. In the second experiment, the population size for *GA*$_1$, *GA*$_2$, *GA*$_3$, *GA*$_4$, *GA*$_5$, *GA*$_6$ was 100. The number of generations was 500. All *GA*s populations used tournament selection of size 3 and crossover rate of 0.5. The mutation rates for center level change, center move, greedy edge mutation, and subtree optimize mutation were 0.7, 0.2, 0.8, and 0.5 respectively.
Each system was allocated 20 runs for each problem instance. All the programs were run on a machine with Pentium 4 Centrino 3.06 GHz CPU using 512MB RAM.

5.4 Results of computational experiments

The experiment shows that:

- Figure 3, 4, 5, 6, 7, 8, 9, 10, 11 show that the proposed heuristic algorithm, called *CBRC* have the best result than *RGH*, *OTTC*, *RGH*$_1$. It means that the solution found by *CBRC* algorithm is the best solution in comparison with the other known heuristic algorithm for solving *BDMST* problem on all the instances with n = 100, 250, 500 and 1000 (n is the number of vertices).
- Figure 15 shows that the best solution found by *GA*$_1$ have better result about 22% than the *CBRC* which is used for initialization the population in *GA*$_1$ on all 20 problem instances.
- Figure 16 shows that sum up of the best solution found by *GA*$_2$ have better result about approximately four times than the *OTTC* which is used for initialization the population in *GA*$_2$ on all 20 problem instances.
- Figure 17 shows that sum up of the best solution found by *GA*$_3$ have better result about over 10 times than the *RGH*$_1$ which is used for initialization the population in *GA*$_3$ on all 20 problem instances.
- Figure 11 shows that sum up of the best solution found by *CBRC* have better result about 6.5 times than the *OTTC* and 17 times than *RGH*$_1$ while the the figure 18 shows that sum up of the best solution found by *GA*$_1$ have better result about 0.8% times than the *GA*$_2$ and approximately 2% than *GA*$_3$.

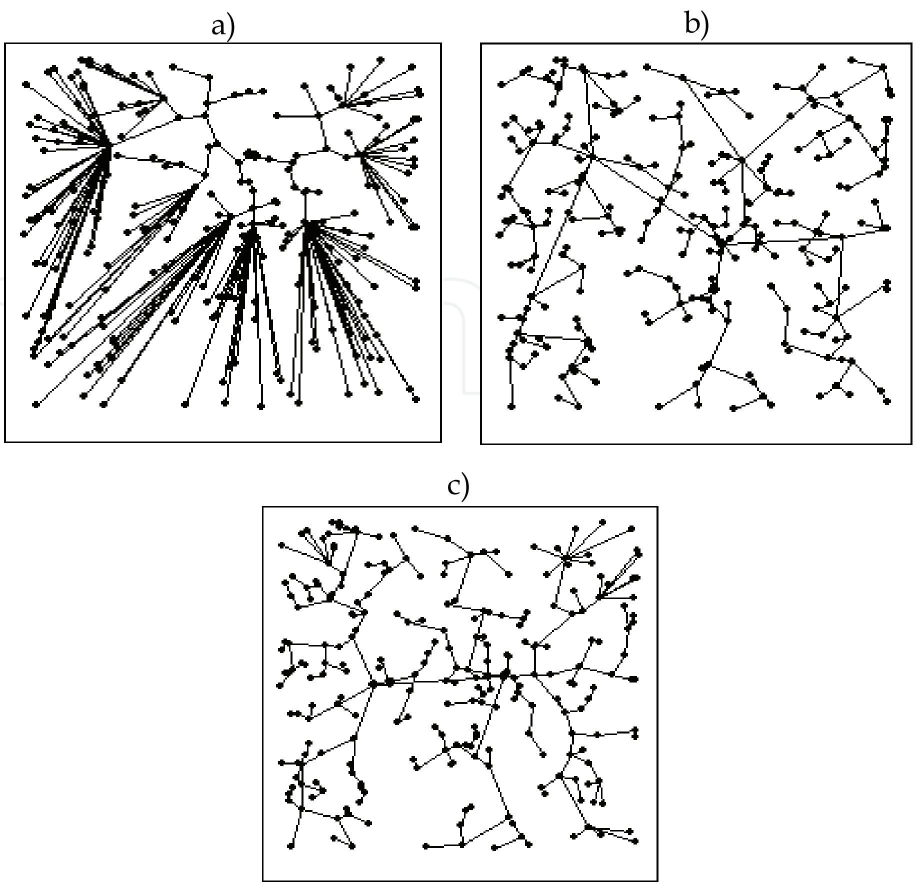

Fig. 3. The best solution found by the heuristics: *OTTC*, *RGH* and *CBRC* on the problem instance with n = 250 and k = 15, test 1:

(a) *OTTC*, weight=42.09; (b) *RGH*, weight=15.14; (c) *CBRC*, weight = 13.32.

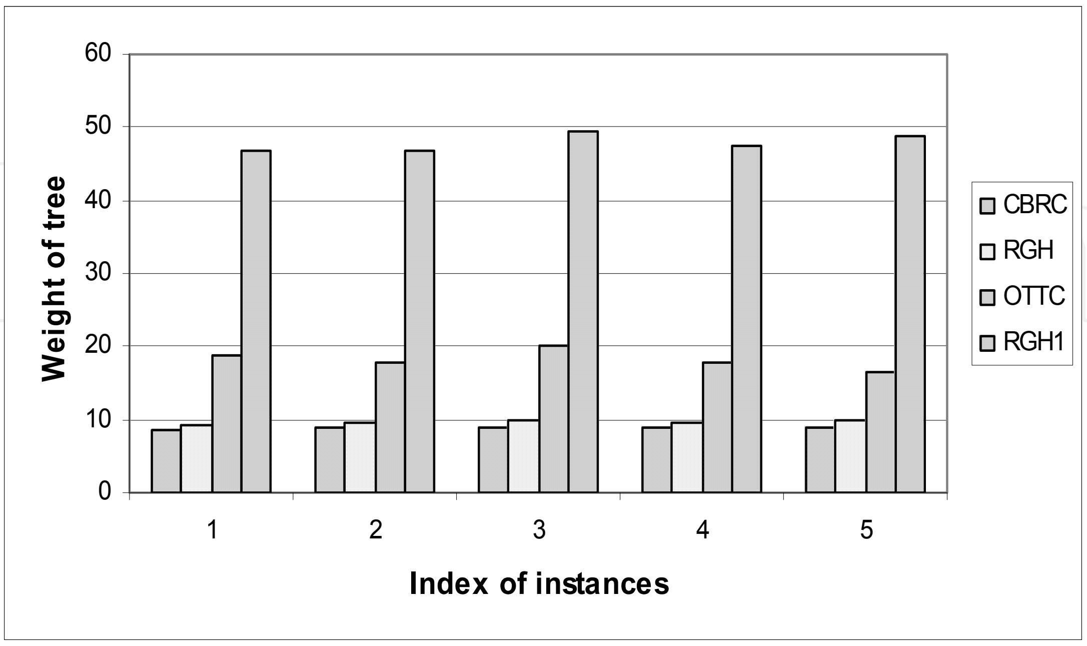

Fig. 4. The best solution found by the four heuristics: *CBR*C, *RGH*, *OTT*C, *RGH*$_1$ on the problem instance with n = 100 and k = 10.

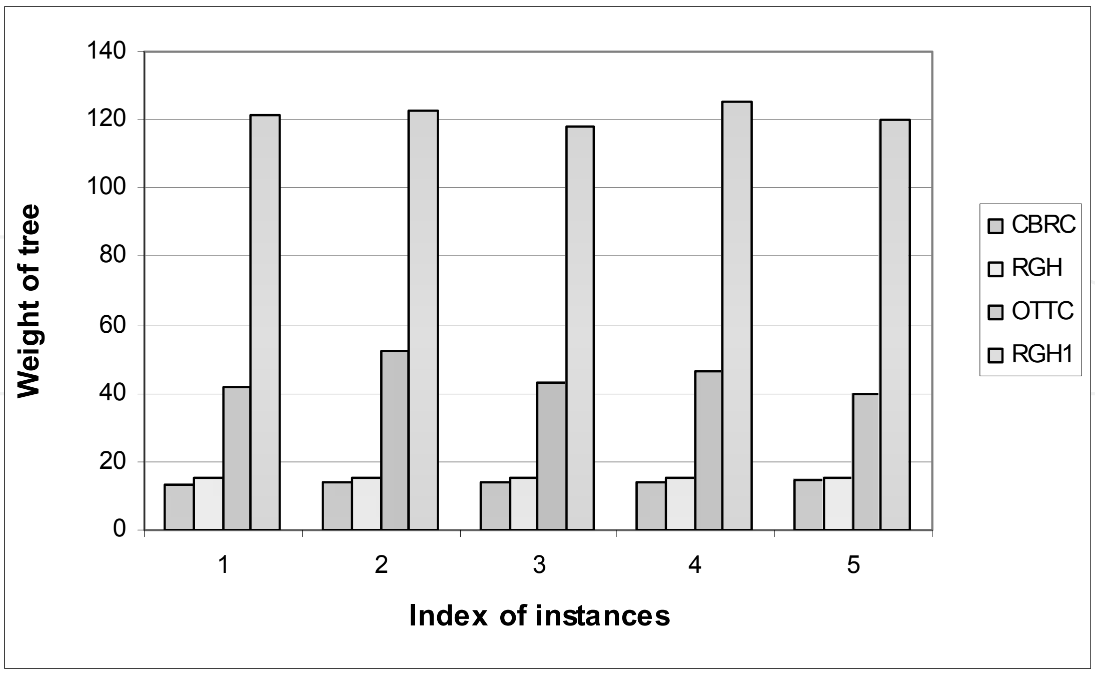

Fig. 5. The best solution found by the four heuristics: *CBRC*, *RGH*, *OTTC*, RGH_1 on the problem instance with n = 250 and k = 15

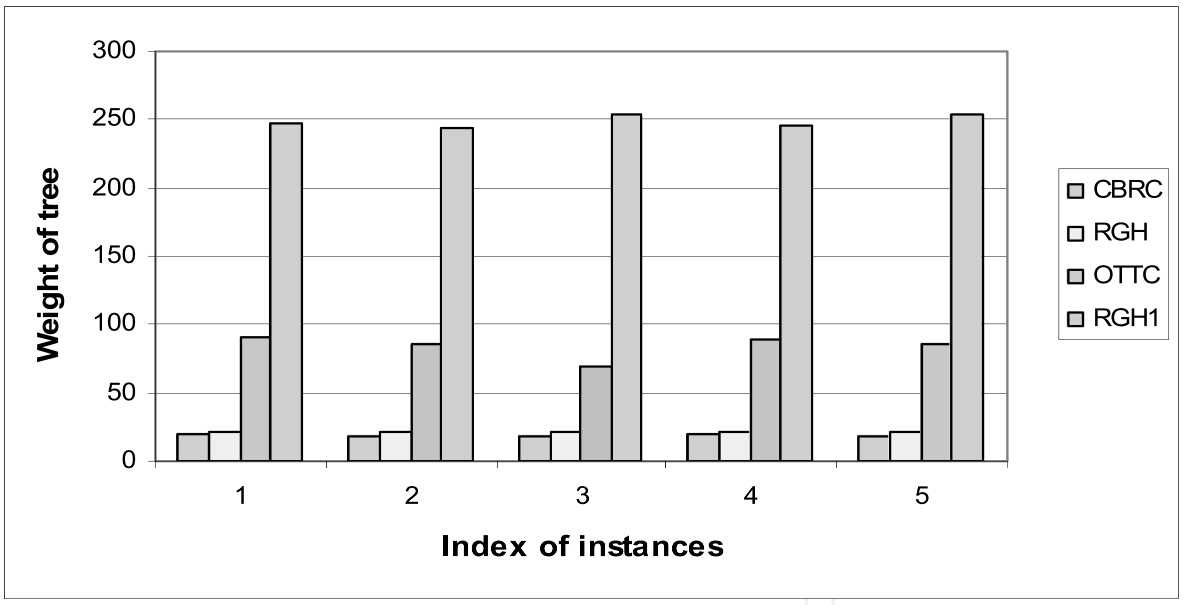

Fig. 6. The best solution found by the four heuristics: *CBRC*, *RGH*, *OTTC*, RGH_1 on the problem instance with n = 500 and k = 20.

- Figure 18 shows that among GA_1, GA_2, GA_3, GA_4, GA_5, GA_6, sum up of the best solution found by GA_6 have bettest result than the other, otherwise GA_3 have worest result.
- Figure 19 shows that GA_1 have smallest sum of standard deviation otherwise GA_3 have largest sum of standard deviation.
- Figure 20 shows that among GA_1, GA_2, GA_3, GA_4, GA_5, GA_6, the number of instances found best result by GA_5 and GA_6 are biggest otherwise the number of instances found best result by GA_2 and GA_3 are smallest.

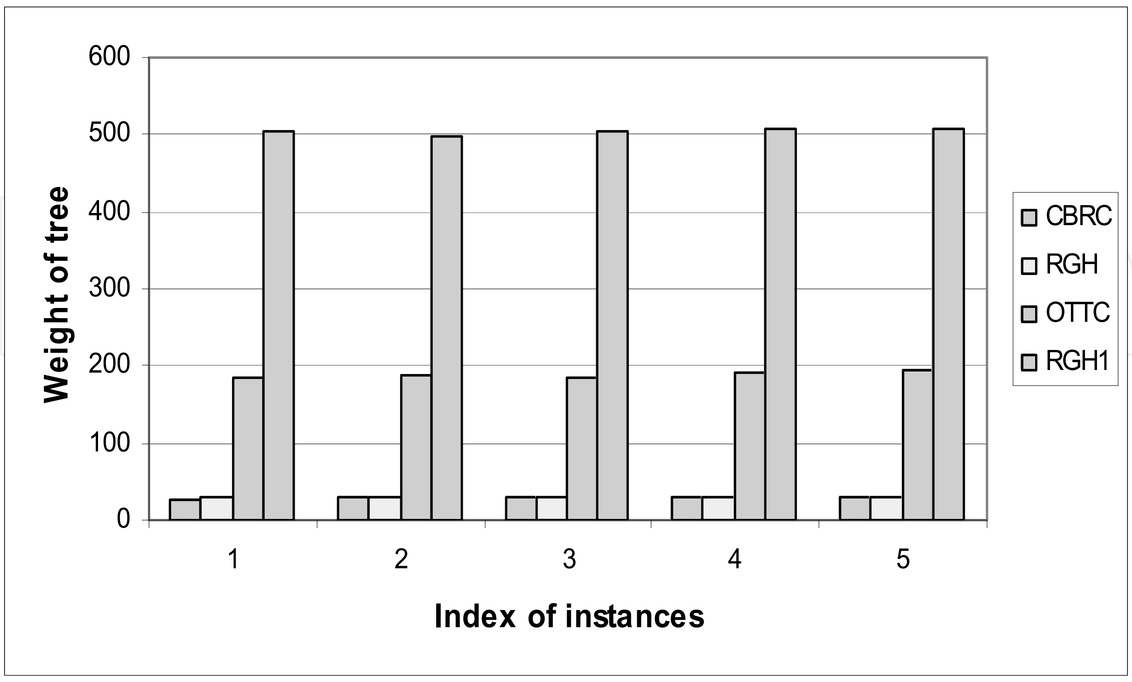

Fig. 7. The best solution found by the four heuristics: *CBRC*, *RGH*, *OTTC*, *RGH*$_1$ on the problem instance with n = 1000 and k = 25.

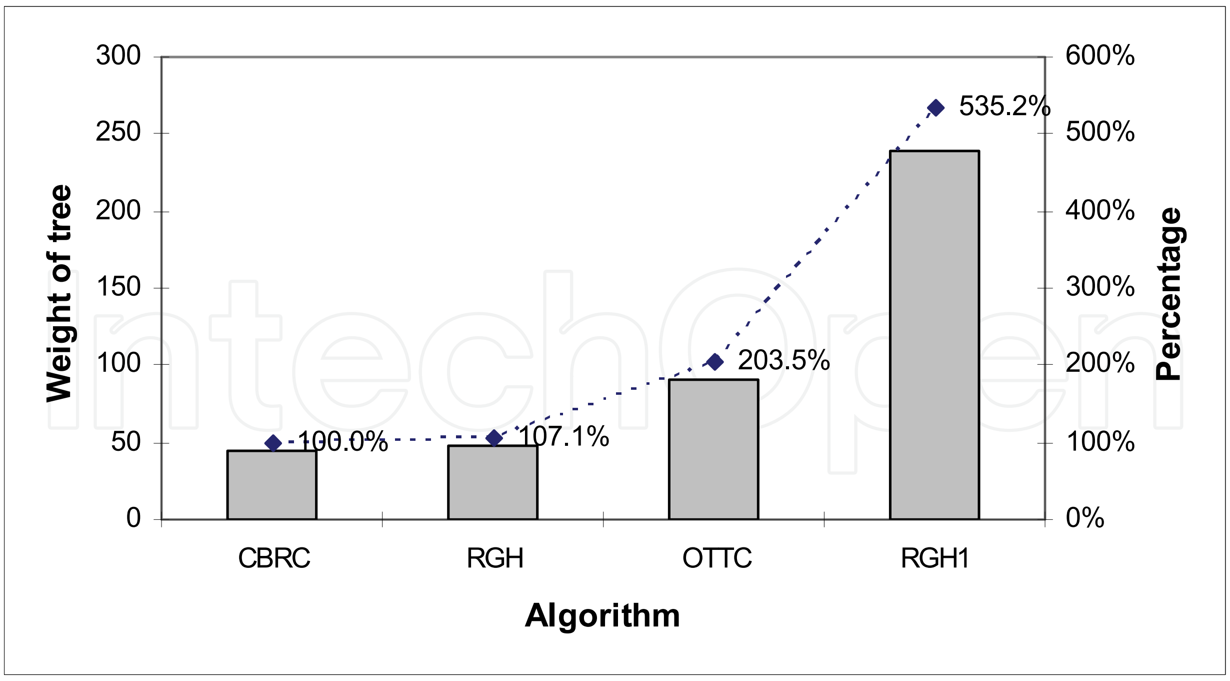

Fig. 8. Comparision of the best solution found by the four heuristics: *CBRC, RGH, OTTC, RGH*$_1$ on all the problem instance with n = 100 (5 instances), k = 10

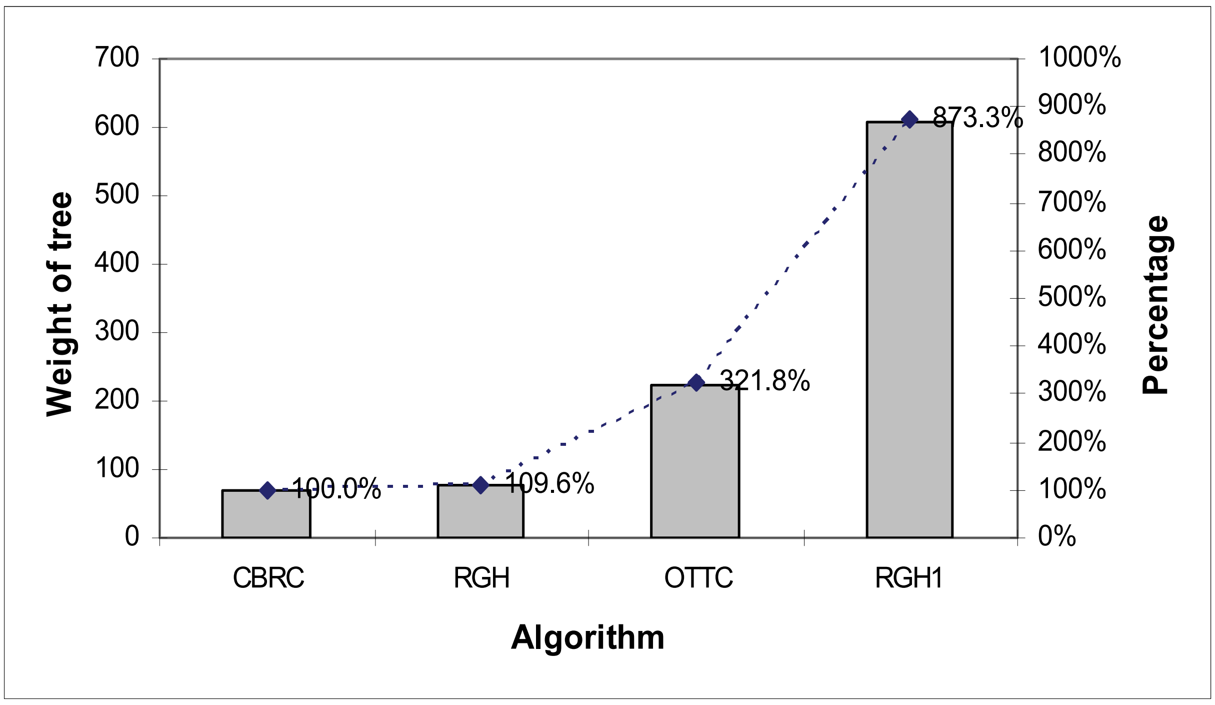

Fig. 9. Comparision between the best solution found by the four heuristics: *CBRC*, *RGH*, *OTTC*, *RGH*$_1$ on all the problem instance with n = 250 (5 instances), k = 15

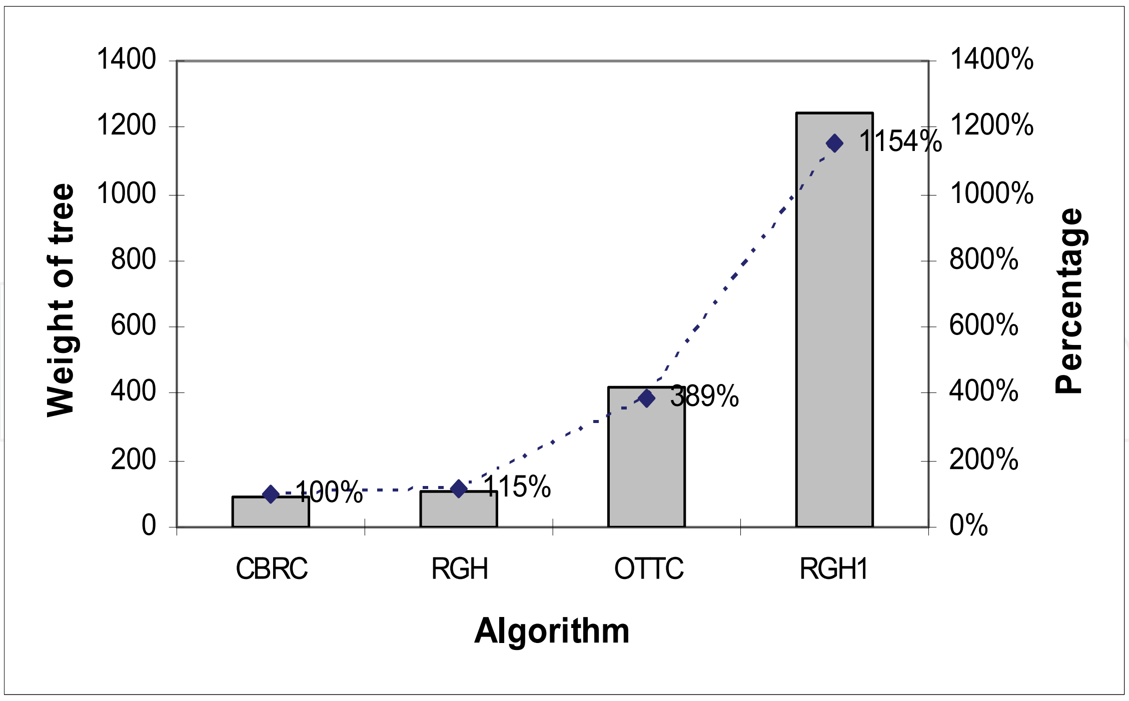

Fig. 10. Comparision between the best solution found by the four heuristics: *CBRC*, *RGH*, *OTTC*, *RGH*$_1$ on all the problem instance with n = 500 (5 instances) , k = 20

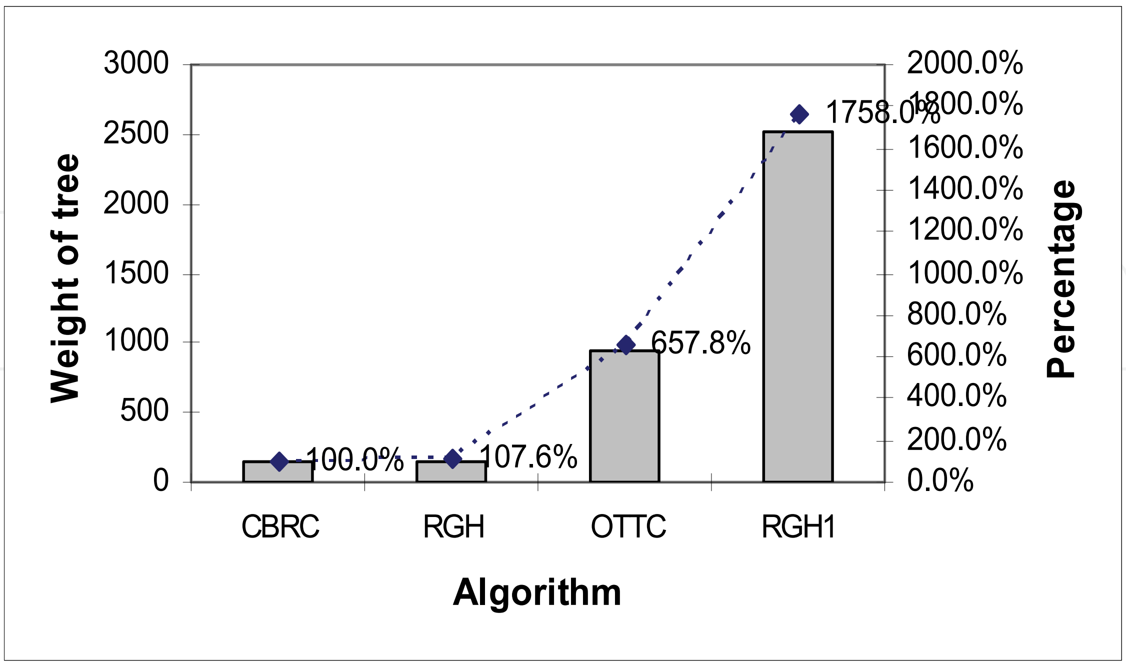

Fig. 11. Comparision between the best solution found by the four heuristics: *CBRC, RGH, OTTC, RGH_1* on all the problem instance with n = 1000 (5 instances) , k = 25

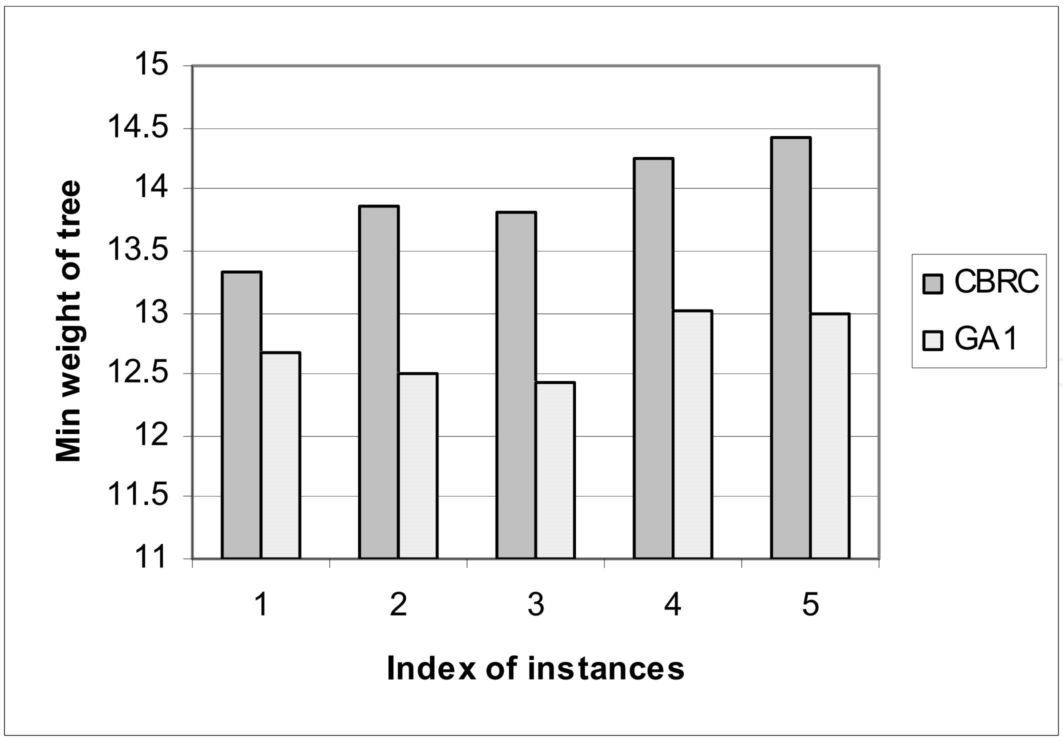

Fig. 12. The best solution found by the *CBRC* and *GA_1* on all the problem instance with n = 250, k = 15

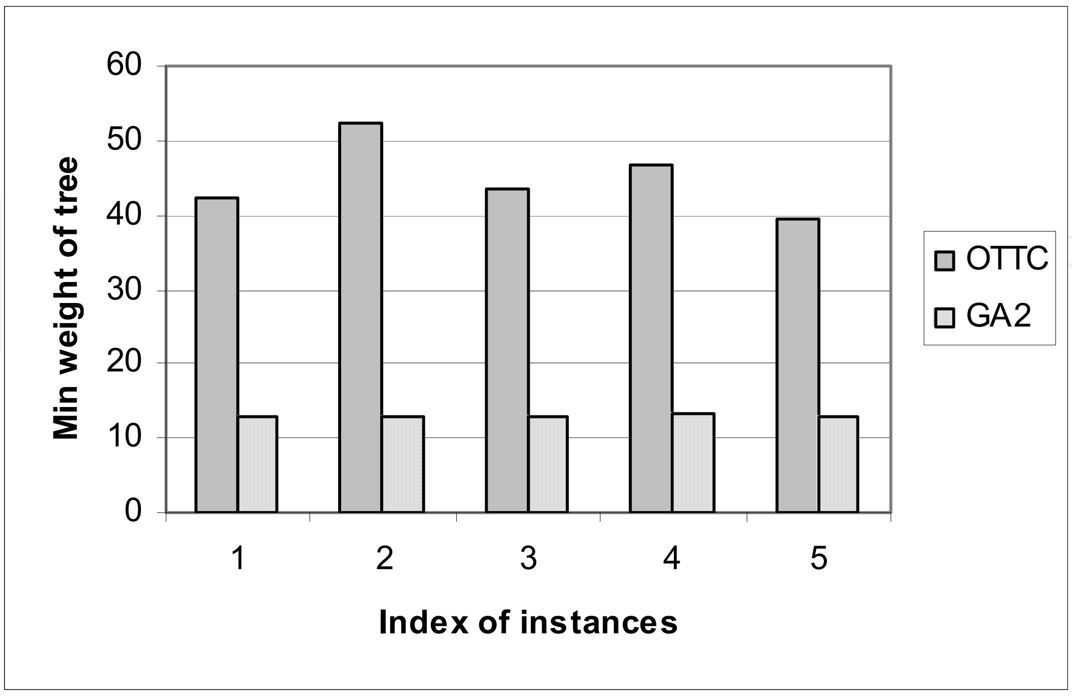

Fig. 13. The best solution found by the *OTTC* and GA_2 on all the problem instance with n = 250, k = 15

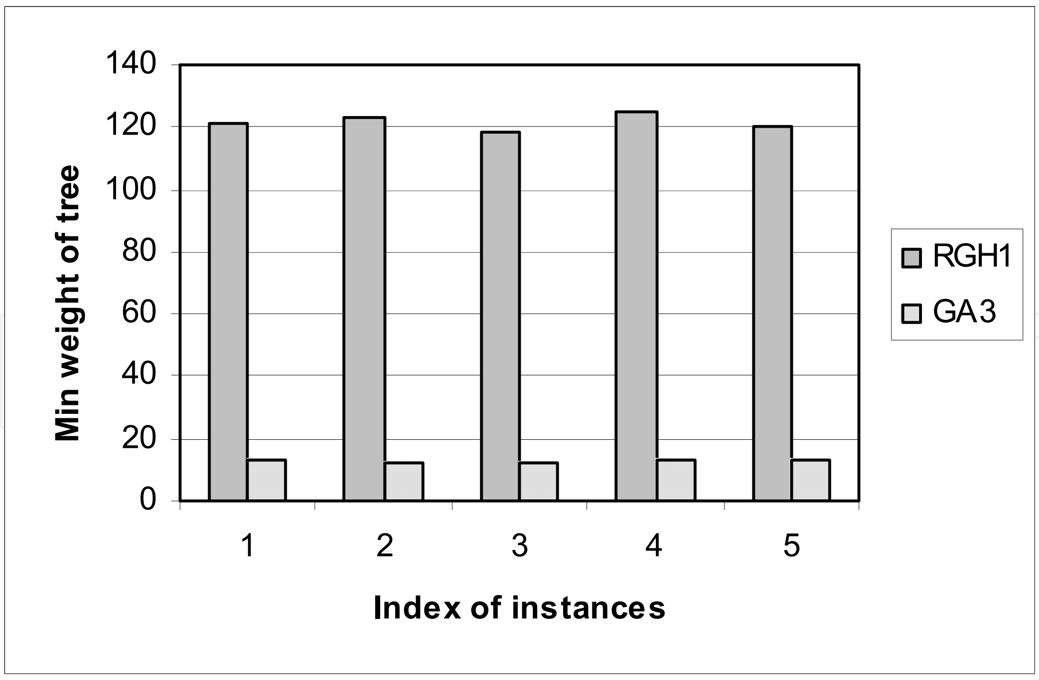

Fig. 14. The best solution found by the RGH_1 and GA_3 on all the problem instance with n = 250, k = 15

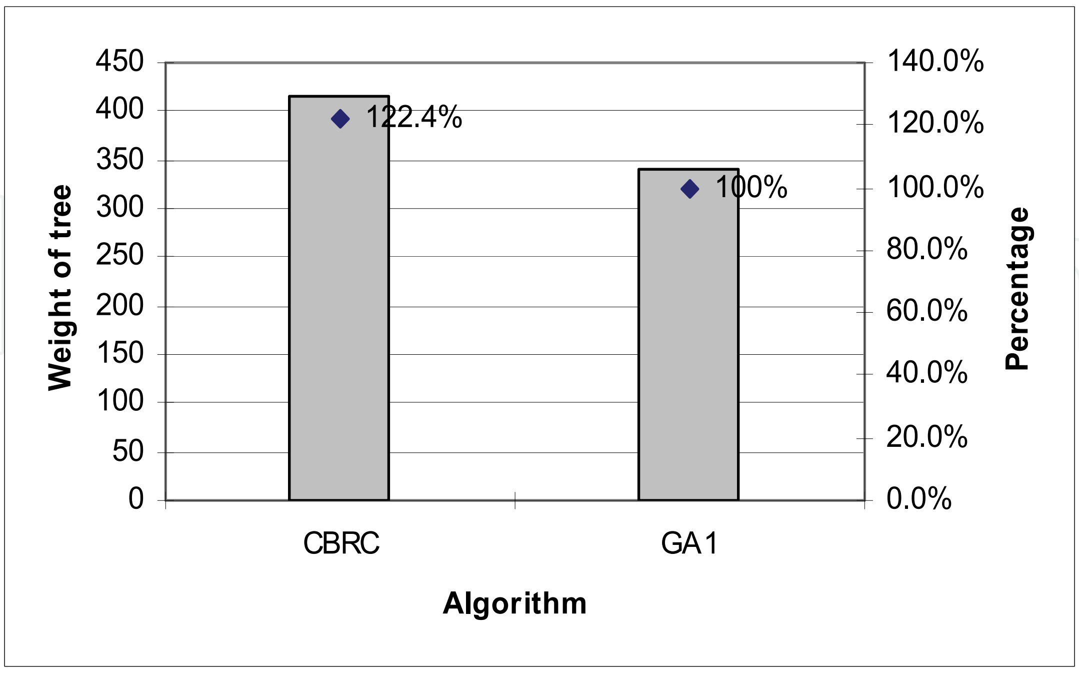

Fig. 15. Sum up the best solution found by the *CBRC* and *GA_1* on all the problem instances (20 instances)

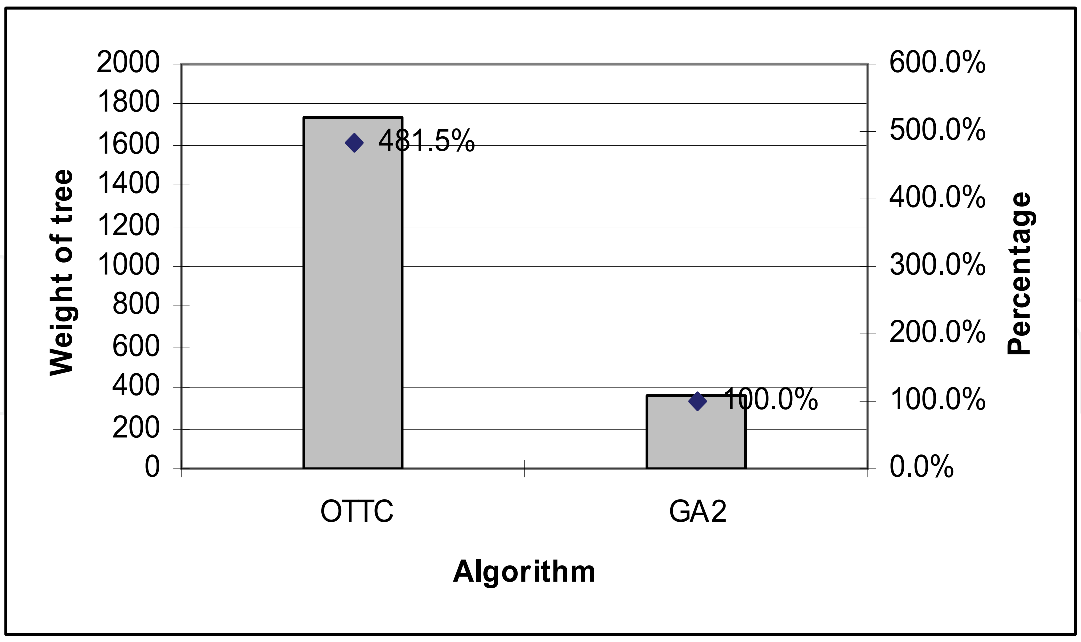

Fig. 16. Sum up the best solution found by the *OTTC* and *GA_2* on all the problem instances (20 instances)

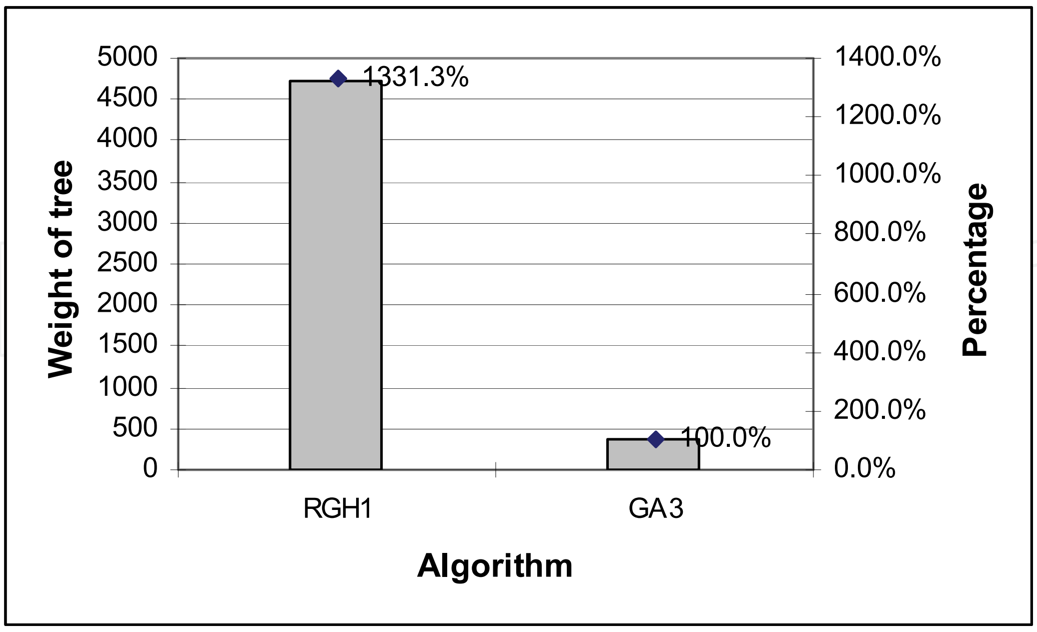

Fig. 17. Comparision between the best solution found by the *RGH*$_1$ and *GA*$_3$ on all the problem instances (20 instances)

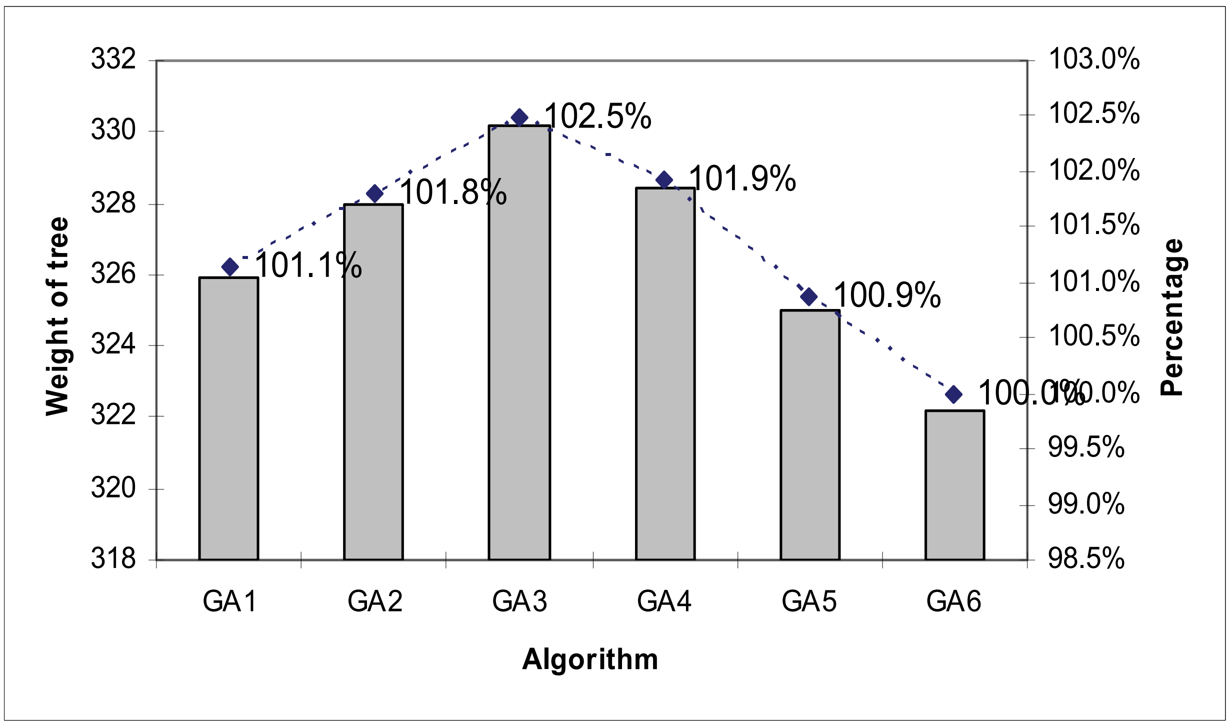

Fig. 18. Comparision between the best solution found by found by *GA*$_1$, *GA*$_2$, *GA*$_3$, *GA*$_4$, *GA*$_5$, GA$_6$ on all the problem instance (20 instances)

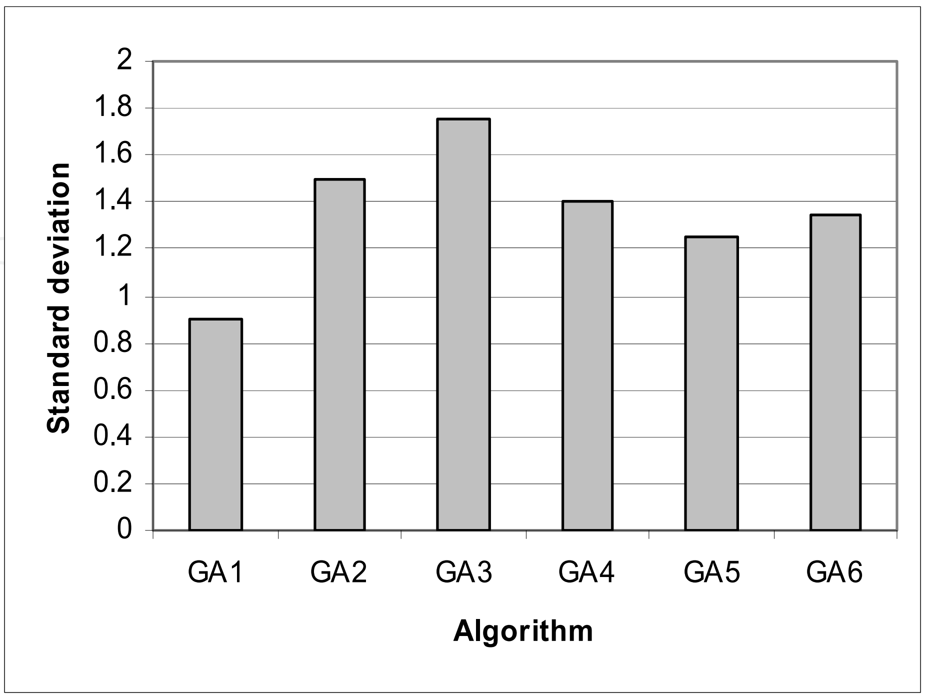

Fig. 19. Comparision between the standard deviation of the solution found by GA_1, GA_2, GA_3, GA_4, GA_5, GA_6 on all the problem instance (20 instances)

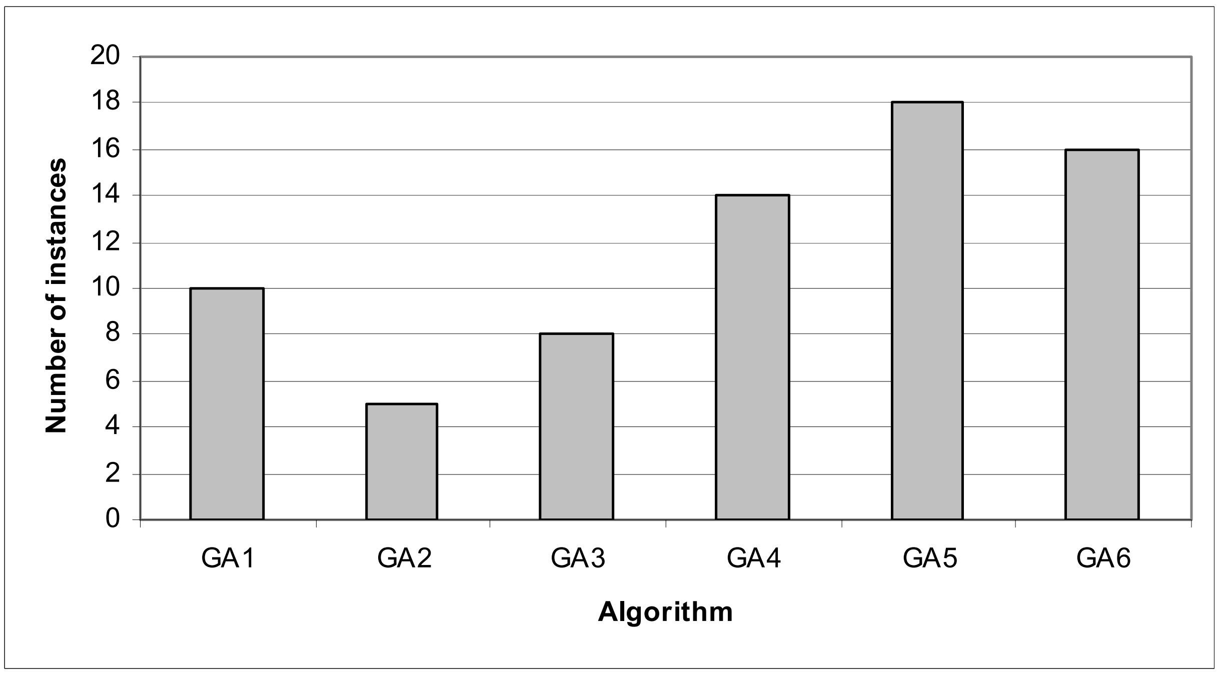

Fig. 20. Number of instances found best result by GA_1, GA_2, GA_3, GA_4, GA_5, GA_6 on all the problem instance (20 instances)

6. Conclusion

We have introduced the heuristic algorithm for solving *BDMST* problem, called CBRC. The experiment shows that *CBRC* have best result than the other known heuristic algorithm for solving *BDMST* prolem on Euclidean instances. The best solution found by the genetic algorithm which uses best heuristic algorithm or only one heuristic algorithm for initialization the population is not better than the best solution found by the genetic algorithm which uses mixed heuristic algorithms (randomized heuristic algorithm and greddy randomized heuristic algorithm). The solution found by the genetic algorithm which uses mixed heuristic algorithm for initialization always is the best result.

7. References

M.R. Garey and D.S.Johnson (1979), Computers and Intractability: A Guide to the Theory of NP-Completeness.

K. Raymond (1989), “A Tree-based Algorithm for Distributed Mutual Exclusion”, ACM Transactions on Computer Systems, 7 (1), 1989, pp. 61-77.

K. Bala, K. Petropoulos (1993), and T. E. Stern, “Multicasting in a linear Lightwave Network”, in Proceedings of IEEE INFOCOM’93, 1993, pp. 1350–1358

C.C. Palmer and A. Kershenbaum (1994), “Representing Trees in Genetic Algorithms”, in Proceedings of The First IEEE Conference on Evolutionary Computation, pp. 379-384

N.R.Achuthan, L.Caccetta, P.Caccetta, and A. Geelen (1994), “Computational Methods for the Diameter Restricted Minimum Weight Spanning Tree Problem”, Australian Journal of Combinatorics, 10, pp.51-71.

NA. Bookstein and S. T. Klein (1996), « Compression of Correlated Bit-Vectors”, Information Systems, 16 (4), pp. 387-400.

G. Kortsarz and D. Peleg (1997), “Approximating Shallow-light Trees”, in Proceedings of the 8th Symposium on Discrete Algorithms, pp. 103-110.

A. Abdalla, N. Deo, and P. Gupta (2000), “Random-tree Diameter and the Diameter Constrained MST”, in Proceedings of Congress on Numerantium, pp. 161-182.

J.Gottlieb, B.A.Julstrom, F.Rothlauf, and G.R.Raidl (2001), “Prufer Numbers: A Poor Representation of Spanning Trees for Evolutionary Search”, in Proceedings of the Genetic and Evolutionary Computation Conference (GECCO’2001).

A. Abdalla (2001), “Computing a Diameter-constrained Minimum Spanning Tree”, PhD Dissertation, The School of Electrical Engineering and Computer Science, University of Central Florida.

G.R. Raidl and B.A. Julstrom (2003), “Edge-sets: An Effective Evolutionary Coding of Spanning Trees”, IEEE Transactions on Evolutionary Computation, 7, pp.225-239.

G.R. Raidl and B.A. Julstrom, (2003) “Greedy Heuristics and an Evolutionary Algorithm for the Bounded-Diameter Minimum Spanning Tree Problem”, in Proceeding of the ACM Symposium on Applied Computing, pp. 747-752.

B.A. Julstrom, G.R. Raidl (2003), “A Permutation Coded Evolutionary for the Bounded Diameter Minimum Spanning Tree Problem, in Proceedings of the Genetic and Evolutionary Computation Conference (GECCO’2003), pp.2-7.

B.A. Julstrom (2004), “Encoding Bounded Diameter Minimum Spanning Trees with Permutations and Random Keys”, in Proceedings of Genetic and Evolutionary Computational Conference (GECCO’2004).

L Gouveia, T.L. Magnanti and C. Requejo (2004), “A 2-Path Approach for Odd Diameter Constrained Minimum Spanning and Steiner Trees”, Network, 44 (4), pp. 254-265.

M. Gruber and G.R. Raidl (2005), “A New 0-1 ILP Approach for the Bounded Diameter Minimum Spanning Tree Problem, in Proceedings of the 2nd International Network Optimization Conference.

M. Gruber and G.R. Raidl (2005), “Variable Neighbourhood Search for the Bounded Diameter Minimum Spanning Tree Problem, in Proceedings of the 18th Mini Euro Conference on Variable Neighborhood Search, Spain.

M. Gruber, J. Hemert, and G.R. Raidl (2006), “Neighbourhood Searches for the Bounded Diameter Minimum Spanning Tree Problem Embedded in a VNS, EA and ACO”, in Proceedings of Genetic and Evolutionary Computational Conference (GECCO’2006).

F. Rothlauf (2006), Representations for Genetic and Evolutionary Algorithms, 2nd Edition, Springer-Verlag.

Nguyen Duc Nghia and Huynh Thi Thanh Binh (2007), “A New Recombination Operator for Solving Bouded Diameter Minimum Spanning Tree Problem”, in Proceedings of RIVF’2007, LNCS.

A. Singh and A.K. Gupta (2007), “An Impoved Heuristic for the Bounded Diameter Minimum Spanning Tree Problem, Journal of Soft Computing, 11, pp. 911-921.

G. Kortsarz and D. Peleg (1999), ”Approximating the Weight of Shallow Steiner Trees”, Discrete Application Mathematics, 93, 1999, pp. 265-285.

R. Prim (1957), “Shortest Connection Networks and Some Generalization”, Bell System Technical Journal, 36, pp. 1389-1401.

Huynh Thi Thanh Binh, Nguyen Xuan Hoai, R.I Ian McKay (2008a), “A New Hybrid Genetic Algorithm for Solving the Bounded Diameter Minimum Spanning Tree Problem”, Proceedings of IEEE World Congress on Computational Intelligence, Hong Kong, LNCS

Huynh Thi Thanh Binh, Nguyen Xuan Hoai, R.I Ian McKay, Nguyen Duc Nghia (2008b), “A new heuristic for the bouded diameter minimum spanning tree problem : some preliminary results”, Conference on Artificial Intelligence PRICAI 2008, submitted

ISBN 978-953-7619-27-5

Hard cover, 586 pages

Publisher InTech

Published online 01, November, 2008

Published in print edition November, 2008