Noname manuscript No.
(will be inserted by the editor)

General Variable Neighborhood Search for the Minimum Stretch Spanning Tree Problem

Yogita Singh Kardam · Kamal Srivastava · Rafael Martí

Received: date / Accepted: date

Abstract Given an undirected graph, the Minimum Stretch Spanning Tree Problem (MSSTP) deals with finding a spanning tree such that the maximum distance in the tree for adjacent nodes in the original graph, called the stretch, is minimum. This is an NP-hard problem with many applications in transportation and communication networks. We propose a General Variable Neighborhood Search (GVNS) algorithm based on a balance between solution generation and improvement. To achieve this balance, we consider different construction heuristics and neighborhood strategies to efficiently explore the search space. To assess the merit of our proposal, we perform extensive experimentation on various classes of graphs consisting of 214 instances. A comparison in terms of solution quality and execution time with the best previous method, namely an Artificial Bee Colony (ABC) algorithm, shows the superiority of GVNS. Results are compared using statistical tests to draw significant conclusions.

Keywords General Variable Neighborhood Search · Artificial Bee Colony · Spanning trees · Minimum Stretch Spanning Tree Problem

1 Introduction

Minimum Stretch Spanning Tree Problem (MSSTP) is based on two well-known problems, finding a spanning tree of a given graph, and finding a shortest path between pairs of nodes in a graph. The former has significance in

Kamal Srivastava
Department of Mathematics, Dayalbagh Educational Institute, Agra, India
E-mail: kamal.sri@dei.ac.in

Yogita Singh Kardam
Department of Mathematics, Dayalbagh Educational Institute, Agra, India

Rafael Martí
Department of Statistics and Operations Research, University of Valencia, Valencia, Spain

networking problems [1, 2, 3] whereas the latter is relevant in operations research, transportation, and VLSI design [4, 5, 6, 7]. A spanning tree of a graph is a subgraph that includes all the vertices in the original graph and is a tree (i.e., a connected acyclic graph). The MSSTP consists of finding a spanning tree of a graph in which the maximum distance in the tree for adjacent nodes in the original graph is minimized.

Given an undirected connected graph $G = (N, A)$, where N is the set of nodes, with $|N| = n$, and $A \subseteq \{(u, w) : u, w \in N\}$ is the set of arcs, the MSSTP is formally defined as follows:

Let

$$\Theta(G) = \{S\_T : S\_T \text{ } is \text{ } spanning \text{ } tree \text{ } of \text{ } G\}$$

Then, MSSTP consists in finding a spanning tree $S\_T^* \in \Theta(G)$ such that

$$Stretch(G, S\_T^*) = \min_{\forall S\_T \in \Theta(G)} \{Stretch(G, S\_T)\}$$

where,

$$Stretch(G, S\_T) = \max_{\forall (u,w) \in A} D_{S\_T}(u, w)$$

and $D_{S\_T}(u, w)$ is the distance (path length) between u and w in S_T.

Let $(u, w) \in A$, then a path between the nodes u and w in S_T is critical if $D_{S\_T}(u, w)$ is maximum over all pairs of adjacent nodes of G; i.e., $D_{S\_T}(u, w) = Stretch(G, S\_T)$. Note that a spanning tree can have more than one critical path. Throughout this paper, a solution S_T to the problem is a spanning tree of the input graph G, and $Stretch(S\_T)$ or simply *Stretch* refers to its objective value.

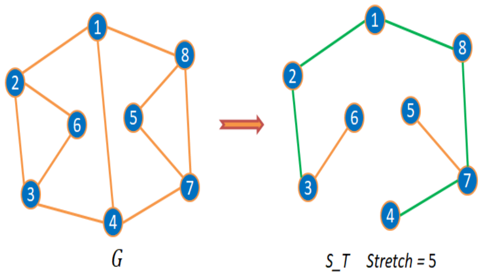

Fig. 1 *Stretch* in spanning tree S_T of a graph G

Figure 1 shows the *Stretch* in the spanning tree S_T of a given graph G. Here, $D_{S\_T}(1, 4) = 3$, $D_{S\_T}(2, 6) = 2$, $D_{S\_T}(3, 4) = 5$, $D_{S\_T}(5, 8) = 2$ and for the remaining arcs of G it is 1. Since the *Stretch* is the maximum distance between two adjacent nodes in G, then in this example the $Stretch = 5$, and the critical path is $(3, 2, 1, 8, 7, 4)$ (shown with green color arcs).

It is worth to mention that the MSSTP is a particular case of the tree t-spanner problem, which involves arc weighted graphs. This problem has applications in various areas such as distributed systems, parallel machine architectures, and communication networks [9]. As an example [10], consider a distributed network of processors in which it is required to route a message between any two processors. This system can be represented as a graph by considering the processors in a network as nodes and the links between them as arcs. As one of the factors on which the cost of routing a message depends is number of arcs between the source and destination, the length of the path (number of arcs) along which the messages traverse has to be minimized. Finding an optimal solution to this problem becomes too expensive for large sized systems. Hence, the problem is to find an efficient routing scheme for large scale communication networks such that the routing cost is minimum. The efficiency of a routing scheme can be measured in terms of the stretch factor, which can be defined as the maximum ratio between the length produced by the routing scheme and the shortest path between the two processors.

The tree t-spanner problem has been extensively studied, but only a few approaches are proposed for the MSSTP. The tree t-spanner was initially defined for constructing network synchronizers [16] in distributed systems, network design and communication networks [17]. Graph theoretic, algorithmic and complexity issues pertaining to tree spanners are studied in [18], and a mixed integer programming formulations with a branch and cut in [19]. In terms of metaheuristics, we can find a Genetic Algorithm (GA) and an ABC algorithm for the weighted tree t-spanner problem [9], where the ABC obtains better results than the GA. Thus, we adapted this ABC for the MSSTP to compare our algorithm over a wide spectrum of instances.

It is a well established area of research in many optimization problems to find optimal solutions for special classes of graphs. That is also the case of the MSSTP, where optimal results have been proved for some specific graphs, such as Peterson graph, complete k-partite graphs, split graphs and rectangular grids [23]. A recent addition to this work includes optimal results for hypercubes, cartesian product of different classes of graphs, hamming graphs and higher-dimensional grids [24, 25]. Some approximation algorithms have also been developed for the problem. In [26], an algorithm which computes a spanning tree with *Stretch* $O(opt^4)$ in time $O(n \log n)$ is developed for the special case of grids and unit disk graphs. However, the scope of our approach is completely different. In particular, we develop a general solving method that can be applied to any instance of this NP-hard problem. Our approach is based on a well-known metaheuristic methodology, the GVNS [27]. This method is based on its efficient search of the solution space in changing the pre-defined neighborhoods in a systematic manner [12]. To the best of our knowledge no other metaheuristic for general graphs has been previously proposed for the MSSTP.

The contribution of this paper is twofold, since we propose a heuristic to outperform the best previous method, and we also propose alternative designs to learn about search strategies in the context of graph problems. In particular, some spanning tree generating procedures are adapted to the MSSTP which serve as the five construction heuristics for generating initial solutions. The VNS metaheuristic is primarily based on a systematic exploration and exploitation of neighborhoods, therefore strategies balancing between diversification and intensification are developed which form our six different neighborhoods.

The remaining paper is organized as follows. Section 2 presents the proposed GVNS and implementation details of its components for MSSTP. The adapted version of ABC algorithm for MSSTP is given in Section 3. The results obtained by GVNS algorithm and two variants of the ABC algorithm for MSSTP over a large range of graphs have been discussed in Section 4. Section 5 provides the conclusions of the paper.

2 General Variable Neighborhood Search

The GVNS proposed for the MSSTP is sketched in Algorithm 1. It starts by generating an initial solution S_T (Step 2) using a construction heuristic described in Section 2.1. The S_T_{best} maintains the best solution found at any step of the algorithm. Step 7 performs the Shake procedure. It is done by randomly generating a neighbor S_T' of S_T in NBHD$_i$ (described in Section 2.2) of S_T. In Step 8, a local minimum solution S_T'' is obtained from S_T' using the Variable Neighborhood Descent (B-VND) method [11]. S_T_{best} is updated if the *Stretch* of S_T'' is better than that of S_T_{best} (Steps 9-11). Now, *Stretch* of the two solutions S_T and S_T'' are compared (Step 12) and S_T is replaced with S_T'' if it improves S_T (Step 13). In this case, i is set to 1 (Step 14) i.e. the new solution will be explored starting again with the first neighborhood. If S_T'' fails to improve S_T in the current neighborhood, then the search is moved to the next neighborhood (Step 16). Steps 7-17 are repeated until all the neighborhoods (1 to nbd_{max}) are explored. The search continues until the stopping criterion is met i.e. $iter_{init}$ reaches the maximum number of iterations $iter_{max}$. We refer the reader to the excellent chapters on VNS [28] and VND [29] in the Handbook of Heuristics [30] for a detailed description about these two methodologies.

Algorithm 2 outlines the procedure B-VND used in GVNS to find a local minimum after exploring all the neighborhoods of a given solution. It starts by finding a best neighbor S_T_1' of solution S_T_1 in its j-th neighborhood using the function *Find_Best_Nbr* (Step 3). Then, neighborhood is changed accordingly by comparing the solutions S_T_1 and S_T_1' (Steps 4-9). S_T_1 keeps improving in a similar way until all the neighborhoods of S_T_1 are explored.

Algorithm 3 presents the function *Find_Best_Nbr* used in B-VND. In Steps 5-13, neighbor S_T_2'' of the solution S_T_2 in the neighborhood NBHD$_k$ is generated. If S_T_2'' improves S_T_2 then S_T_2'' replaces S_T_2 and the process is repeated in this neighborhood. If no improvement is there then the last obtained solution is compared with the original one. This original solution is replaced by a better solution. The process (Steps 4 to 19) is repeated until no

Algorithm 1 General Variable Neighborhood Search Algorithm for MSSTP (GVNS)

```
Initialize number of neighborhoods (nbd_max) and number of iterations (iter_max)
S_T ← generate initial solution
S_T_best ← S_T
while iter_init ≤ iter_max do
    i ← 1
    while i ≤ nbd_max do
        S_T' ← NBHD_i(S)
        S_T'' ← B-VND(S_T', nbd_max)
        if Stretch(S_T'') < Stretch(S_T_best) then
            S_T_best ← S_T''
        end if
        if Stretch(S_T'') < Stretch(S_T) then
            S_T ← S_T''
            i ← 1
        else
            i ← i + 1
        end if
    end while
    iter_init ← iter_init + 1
end while
return S_T_best
```

Algorithm 2 B-VND(S_T_1, nbd_{max})

```
j ← 1
while j ≤ nbd_max do
    S_T_1' ← Find_Best_Nbr(S_T_1, j)
    if Stretch(S_T_1') < Stretch(S_T_1) then
        S_T_1 ← S_T_1'
        j ← 1
    else
        j ← j + 1
    end if
end while
return S_T_1
```

further improvement is obtained. The different construction heuristics and the neighborhood strategies used in GVNS are discussed below.

2.1 Initial Solution Generation

The initial solution is generated using a construction heuristic selected randomly from the five construction heuristics namely-

- $Random\_Prim$
- $Random\_Kruskal$
- $Random\_Dijkstra's$
- Max_degree_BFS
- $Random\_BFS$

Algorithm 3 $Find\_Best\_Nbr(S\_T_2, k)$

$flag_1 \leftarrow 0$
$flag_2 \leftarrow 0$
while $flag_1 \neq 1$ **do**
    $S\_T_2' \leftarrow S\_T_2$
    while $flag_2 \neq 1$ **do**
        $S\_T_2'' \leftarrow \text{NBHD}_k(S\_T_2)$
        if $Stretch(S\_T_2'') < Stretch(S\_T_2)$ **then**
            $S\_T_2 \leftarrow S\_T_2''$
            $flag_2 \leftarrow 0$
        else
            $flag_2 \leftarrow 1$
        end if
    end while
    if $Stretch(S\_T_2) < Stretch(S\_T_2')$ **then**
        $flag_1 \leftarrow 0$
        $flag_2 \leftarrow 0$
    else
        $flag_1 \leftarrow 1$
    end if
end while
return S_T_2'

$Random\_Prim$ (described in Section 3), $Random\_Kruskal$ and $Random\_Dijkstra's$ construct spanning tree using well known Prim's, Kruskal's and Dijkstra's algorithms [8] respectively by selecting arcs randomly and considering the unit weights on the arcs. The other two heuristics Max_degree_BFS and $Random\_BFS$ produce spanning trees using the well known Breadth First Search (BFS) algorithm. Max_degree_BFS explores all the nodes of the graph starting from a maximum degree node as root. The neighbor nodes are also traversed in decreasing order of their degrees. The process continues until all nodes are traversed. Preferring higher degree nodes helps in keeping the neighbors close and hence may lead to a spanning tree with lower $Stretch$. In $Random\_BFS$ a spanning tree is produced by visiting neighbors randomly.

2.2 Neighborhood Strategies

We have designed six neighborhood strategies for generating a neighbor of a given solution as detailed below.

1. **Method 1 (**NBHD_1**):** In this method neighbors of solutions are generated based on the cycle exchange. An arc $(u, w) \in A(\text{G})\backslash A(S\_T)$ is randomly selected, and added to S_T creating a cycle Cyc. Now, $(u', w') \in Cyc\backslash(u, w)$ is picked up randomly and removed from Cyc resulting in a neighbor S_T'. This method helps in diversification as the arcs to be added and deleted are chosen randomly (see Algorithm 4).
   Figure 2 illustrates this process. An arc $(4, 6)$ belonging to G is added to its spanning tree S_T which forms a cycle in S_T. Now the arc $(4, 7)$ appearing in this cycle is removed from S_T producing a neighbor S_T'.

Algorithm 4 NBHD$_1$ (S_T)

1: select (u, w) randomly from $A(G)\backslash A(S\_T)$
2: $S\_T \leftarrow S\_T \cup (u, w)$
3: $Cyc \leftarrow$ cycle obtained by adding arc (u, w) to S_T
4: $(u', w') \leftarrow$ select randomly such that $(u', w') \in A(Cyc)$
5: $S\_T' \leftarrow S\_T\backslash(u', w')$
6: **return** S_T'

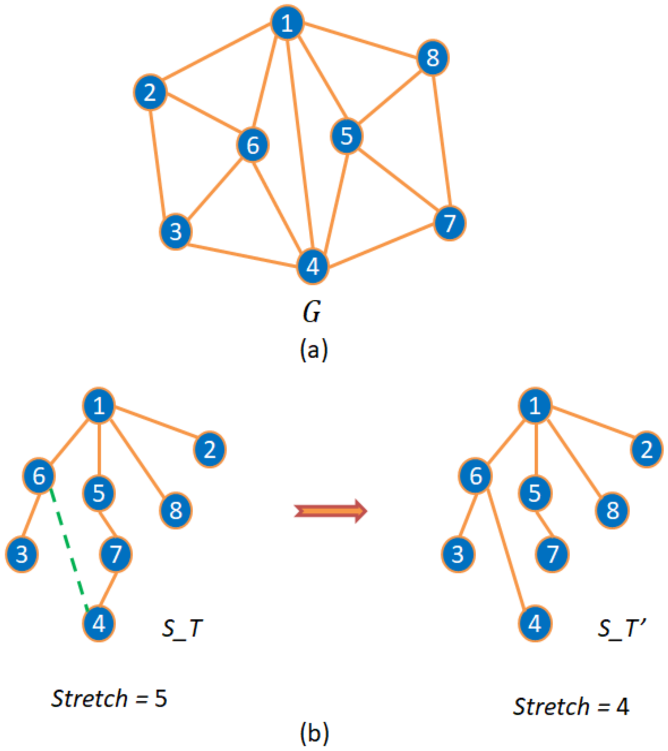

Fig. 2 (a) Graph G and its (b) Spanning tree S_T with its neighbor S_T' obtained from NBHD$_1$

2. **Method 2** (NBHD$_2$)**:** This method generates a neighbor of a spanning tree by replacing one of its subtree with another subtree of the graph (see Algorithm 5). Initially, a critical path C_P in S_T is randomly selected and a subgraph G' of G induced by the nodes of C_P is formed. A spanning tree P_T of G' is then generated using the heuristic $Random\_Prim$ described in Section 2.1. Now with the help of partial tree P_T and the given S_T, a neighbor S_T' is obtained by adding those arcs of S_T to P_T which are not in C_P. This method favors the intensification as one of the critical paths is chosen for the replacement and hence may provide an improved solution.

This procedure is illustrated in Fig. 3. S_T in Fig. 3 (a) shows a spanning tree of G in Fig. 2 (a) which has a critical path $\{(5, 8, 1, 6, 4, 7)$ corresponding to *Stretch* 5. Now, this path (shown with green color arcs) is selected and a subgraph G' of G is produced from its nodes. A spanning tree P_T of G' is created using $Random\_Prim$. This P_T is transformed into a complete spanning tree S_T' by adding those arcs (shown with the green color dotted lines) to it from S_T which are not in $(5, 8, 1, 6, 4, 7)$.

Algorithm 5 NBHD_2 (S_T)

```
1: C_P ← critical path in S_T selected randomly
2: N' ← nodes in C_P
3: A(C_P) ← arcs in C_P
4: G' ← subgraph of G with node set N'
5: P_T ← spanning tree of G' generated using Random_Prim
6: S_T' ← P_T ∪ {(u, w) : (u, w) ∈ A(S_T)\A(C_P) }
7: return S_T'
```

The remaining methods NBHD_3 to NBHD_6 are similar to NBHD_2, where the partial tree P_T is generated using the heuristics $Random\_Kruskal$, $Random\_Dijkstra's$, Max_degree_BFS and $Random\_BFS$ respectively.

3 Artificial Bee Colony Algorithm for MSSTP

The ABC algorithm is the best heuristic identified in the related literature for tree t-spanner problem [9]. We therefore adapted it to the MSSTP. The basic framework of ABC is inspired from the intelligent foraging behaviour of honey bees. Based on their behaviour, the bees are divided into three categories: employed bees, onlooker bees and scout bees. The solutions of the initial population ($pop_{employed}$) refers to the food sources whereas the fitness (quality) of these solutions refer to the nectar amount of the respective food sources. Now these food sources are exploited by generating neighbors with the help of employed bees. The role of onlooker bees is to further exploit those food sources which are rich in nectar amount in order to find the better food sources (the ones containing more nectar). If a food source is not improved by the employed bee for a fixed number of iterations, then it is replaced by a new food source found by the scout bee.

In the context of MSSTP, we first describe the solution generation method followed by the neighborhood strategies of [9]. For a fairer comparison, we have used the strategies of [9] for these two procedures. Prim's algorithm [31] is adapted to construct a solution by selecting arcs randomly rather than choosing them according to their weights. Two neighborhoods nbd_1 and nbd_2 are adapted suitably to locally improve a given solution based on a set of arcs A_c in the critical path. Note that a path from u to w is critical in the case of tree t-spanner problem if $t = Stretch\_Factor(G, S\_T)$ whereas for MSSTP, the critical path is as defined in Section 1.

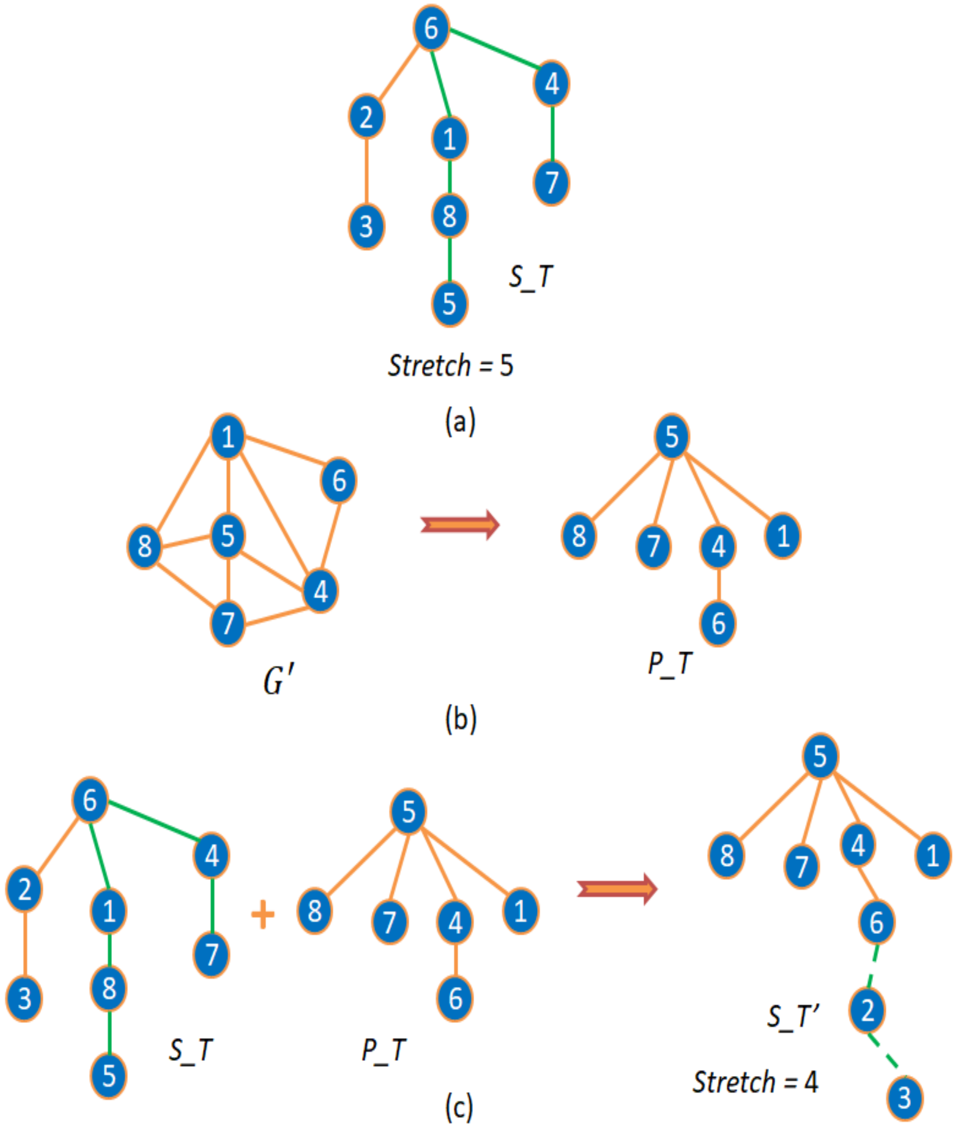

Fig. 3 (a) Spanning tree S_T of G in Fig. 2 (a) and its (b) subgraph G' induced by the vertex set of critical path C_P and its spanning tree P_T obtained using *Random_Prim*, (c) neighbor S_T' of S_T obtained from S_T and P_T using NBHD$_2$

The first local search method is based on the neighborhood nbd_1 (see Algorithm 6) in which first an arc $(u, w) \in A_c$ from a solution S_T_1 is deleted resulting in two disconnected components of S_T_1. Then, a neighbor S_T_1' is produced by joining these components together. For this another solution S_T_2 is selected randomly from the population and an arc from S_T_2, which can join the two components, is found by checking all the arcs iteratively. In the absence of such an arc in S_T_2, the process is repeated with another solution S_T_2 of the population. The second local search method, contrasts with the first one in the way in which the two disconnected components are reconnected. The arcs required for this purpose are picked up from the graph G resulting in a better exploitation of the neighborhood (nbd_2 in this case).

Algorithm 6 nbd_1 (S_T_1)

$C\_P \leftarrow$ critical path in S_T_1 selected randomly
$A_c \leftarrow$ arcs in C_P
Select an arc $(u, w) \in A_c$ randomly
$[P\_T_1, P\_T_2] \leftarrow \{(u_1, w_1) : (u_1, w_1) \in A(S\_T_1) \backslash (u, w)\}$
$joined \leftarrow 0$
while $joined \neq 1$ **do**
    $S\_T_2 \leftarrow$ select a solution $\in pop_{employed}$ randomly
    for each arc $(u', w') \in S\_T_2$ **do**
        if (u', w') can join P_T_1 and P_T_2 **then**
            $S\_T'_1 \leftarrow P\_T_1 \cup P\_T_2 \cup (u', w')$
            $joined \leftarrow 1$
        end if
    end for
end while
return $S\_T'_1$

In the ABC procedure (outlined in Algorithm 7), Prim's algorithm is used to generate m_1 number of initial solutions. Each solution of the population is referred as an employed bee. For the iterative procedure, neighbor of a solution is obtained by probabilistically selecting one of the local search methods. If neighbour is better, it replaces the original solution. If after $iter_{limit}$ number of iterations, the solution does not improve then the scout bee produces a new solution which is used to replace the non-improving solution. A Binary Tournament Selection method (which picks a solution with better fitness from the two randomly selected solutions) is used to select m_2 solutions (acting as onlooker bees) out of m_1 solutions of the population with a probability P_{bts}. This is done with an objective to further exploit their neighborhoods using the two neighborhood strategies of [9]. The complete procedure is repeated until the termination criterion is met.

For further impartial comparison of GVNS with the ABC approach, we also implemented ABC with the neighborhood strategies used in GVNS. We refer this version as ABC_Nbr.

4 Experimental Results and Analysis

The two main objectives of our experimentation consist of investigating different search strategies, and then comparing the performance of the three algorithms ABC, ABC_Nbr and GVNS for MSSTP in terms of solution quality

Algorithm 7 Artificial Bee Colony Algorithm for MSSTP (ABC)

$pop_{employed} \leftarrow$ generate initial solutions $S\_T_i, i = 1$ to m_1 using Prim's algorithm
$S\_T_{best} \leftarrow$ best solution of $pop_{employed}$
while termination criterion is not met **do**
  for $i = 1$ to m_1 **do**
    if $\rho < P_{nbd}$ **then**
      $S\_T_i' \leftarrow nbd_1(S\_T_i)$
    else
      $S\_T_i' \leftarrow nbd_2(S\_T_i)$
    end if
    if $Stretch(S\_T_i') < Stretch(S\_T_i)$ **then**
      $S\_T_i \leftarrow S\_T_i'$
    else
      if S_T_i does not improve till $iter_{limit}$ **then**
        $S\_T_{scout} \leftarrow$ generate solution using Prim's algorithm
        $S\_T_i \leftarrow S\_T_{scout}$
      end if
    end if
    if $Stretch(S\_T_i) < Stretch(S\_T_{best})$ **then**
      $S\_T_{best} \leftarrow S\_T_i$
    end if
  end for
  for $i = 1$ to m_2 **do**
    $S\_T_{o_i} \leftarrow$ select solution from $pop_{employed}$ using Binary Tournament selection method
    if $\rho < P_{nbd}$ **then**
      $S\_T_{onlooker_i} \leftarrow nbd_1(S\_T_{o_i})$
    else
      $S\_T_{onlooker_i} \leftarrow nbd_2(S\_T_{o_i})$
    end if
  end for
  for $i = 1$ to m_2 **do**
    if $Stretch(S\_T_{onlooker_i}) < Stretch(S\_T_{o_i})$ **then**
      $S\_T_{o_i} \leftarrow S\_T_{onlooker_i}$
    end if
    if $Stretch(S\_T_{onlooker_i}) < Stretch(S\_T_{best})$ **then**
      $S\_T_{best} \leftarrow S\_T_{onlooker_i}$
    end if
  end for
end while

and running time. The three algorithms are implemented in C++ on ubuntu 16.04 LTS machine with Intel(R) Core(TM) i5-2400 CPU @3.10×4 GHz and 7.7 GiB of RAM. For the experiments, we consider two sets of instances as our test suite - Set A consists of 10 classes of graphs with known optimal results [23], whereas Set B consists of 70 instances of Harwell- Boeing (HB) graphs taken from the public domain Matrix Market library (available at https://math.nist.gov/MatrixMarket/data/Harwell-Boeing/) that are mostly used for the optimization problems in communication networks. Optimal results are not known for this latter set .

The classes of graphs in Set A are listed in Table 1. In the second column of the table, the size of the graphs (i.e. number of nodes) is shown varying within the specified range. '# **inst**' shows the number of instances generated for each

Table 1 Graphs in Set A

Graphs	size	# inst	optimal
Pt	10	1	4
$K_{1,n-2,1}$	[4, 120]	10	2, for $n \geq 4$
C_n	[5, 150]	10	$n-1$, for $n \geq 3$
W_n	[5, 1500]	18	2, for $n \geq 4$
K_n	[5, 1400]	17	2, for $n \geq 3$
S_n	[10, 1495]	18	-
$K_{n_1,n_2,\ldots,n_k}$	[8, 1500]	17	3, if $k=2, n_1, n_2 \geq 2$ or $k \geq 3, n_1 \neq 1$ 2, if $k \geq 3, n_1 = 1$
T_n	[10, 1485]	17	$\lceil \frac{2n}{3} \rceil + 1$, for $n \geq 1$
$P_m \times P_n$ (small)	[6, 100]	9	$2\lfloor \frac{m}{2} \rfloor + 1$, $2 \leq m \leq n$
$P_m \times P_n$ (large)	[104, 1080]	9	same as for small
$TR_{m,n}$ (small)	[12, 75]	9	m, for $2 \leq m \leq n$
$TR_{m,n}$ (large)	[150, 1500]	9	same as for small

Table 2 Parameter tuning for ABC methods

parameters	values tested for tuning	values used
m_1	25, 50, 100, 150	50
m_2	25, 50, 100, 150	100
P_{bts}	0.75, 0.80, 0.85, 0.90	0.80
P_{nbd}	0.90, 0.95, 0.98	0.95
$iter_{limit}$	50, 100, 150	100

class. The last column gives the optimal values of each of the classes of graphs except Split graphs whose details are given in [23]. For the preliminary experiments, and to avoid the overtraining of the methods, a representative set of 10 HB graphs consisting of $lund\_a, lund\_b, steam1, dwt361, bcsstm07, pores3, dwt592, steam2, fs\_680\_1$ and $saylr3$ instances is considered as the training set.

To carry out the experiments, five independent trials are conducted for each of the three algorithms- ABC, ABC_Nbr and GVNS on each instance of the test suite. Considering that we reimplemented the ABC algorithm to target the MSSTP, we perform a preliminary experiment to adjust its parameters. Table 2 shows the values tested for the five search parameters of the method, and those that achieved the best results in a full-factorial design. We have empirically found that after 50,000 evaluations the ABC methods do not yield further improvements. We therefore set this value as the termination criterion in our experiments.

To investigate the performance of the construction heuristics used in generation of the initial solution in GVNS, a second experiment is conducted on the subset of representative instances. For this, five independent runs of GVNS are performed (a) using Max_degree_BFS (found to be the best construction heuristic among the five heuristics by our preliminary experiments aimed to compare their performance) only and (b) by selecting one of the five

Table 3 Comparison of neighborhood strategies in GVNS

	two_nbd	*all_nbd*
Stretch	15.88	**15.84**
gbest_time	83.49	**31.23**
t_c	428.10	451.37

construction heuristics (see Section 2) randomly. As no significant difference is found in the results obtained, random selection of construction heuristic is implemented in our method to have diverse initial solutions in different runs of GVNS.

The third preliminary experiment is conducted to analyse the performance of the neighborhood strategies used in GVNS to generate neighbors of a given solution. For this, GVNS is run (a) using two neighborhood strategies $NBHD_1$ and $NBHD_2$ only (referred as *two_nbd*) and (b) using all the neighborhood strategies (referred as *all_nbd*). Table 3 shows the average values of *Stretch*, *gbest_time* (time to attain global best solution) and t_c (completion time) over five independent runs obtained by GVNS using both methods. From the results, it can be seen that the performance of GVNS is better when all the neighborhood strategies are incorporated as compared to using only two neighborhoods. Therefore, GVNS with all neighborhood strategies is used for the main experiments.

In our final preliminary experiment we study the order in which the neighborhoods are explored. In particular, we consider five different versions of our GVNS algorithm that only differ in the order in which the neighborhoods are scanned within the variable neighborhood template. We did not observe significant differences in the results and therefore we do not include the results of this experiment.

We now compare our method, set with the key strategies determined above, with the best methods identified in the literature. All the three algorithms give optimal results for Cycle graph, Diamond graph and Peterson graph (classes of Set A). The results of remaining classes obtained by the three algorithms are shown in Table 4. In the table, 'O_a' is the average of known optimal *Stretch* values over the number of instances corresponding to each class. 'S_T_{min}' and 'S_T_{avg}' show the minimum and average *Stretch* values respectively, obtained by the algorithms in 5 runs, while 't_{avg}' shows the average execution time (in seconds) to attain the global best solution over same number of runs. '#best' is the number of best solutions of S_T_{min} among the three algorithms, and '#opt' show the number of optimal values attained by these algorithms for the instances of Set A. It is to be noted here that the results are the averages of the values attained by the algorithms for each parameter (i.e S_T_{min}, S_T_{avg}, t_{avg}, #best and #opt) over the number of instances for each class of graphs. Table 5 presents the results obtained by these algorithms on HB graphs reporting the same parameters (except #opt) as given in Table 4. Following result for general graphs [26] is used to compute the lower bound (LB) for the instances of HB graphs:

$$Stretch(G, S\_T^*) \geq \max_{\forall (u,w) \in A(G)} D'_G(u, w)$$

where, $D'_G(u, w)$ is the shortest distance between u and w in $G \backslash (u, w)$ and S_T^* is an optimal solution. It is observed that for the instances of this class lower bounds (optimal values) are attained by GVNS, ABC, and ABC_Nbr in 9, 7 and 6 cases out of a total of 70 cases respectively. We have categorized HB graphs based on their sizes as small graphs ($|N| \leq 100$), medium graphs ($100 < |N| \leq 500$) and large graphs ($500 < |N| \leq 1000$) for a fairer comparison of mean values. In the tables, the results for which the performance of GVNS is better than that of ABC and ABC_Nbr are shown in bold. These comparisons consistently show that GVNS outperforms the other two methods, since GVNS attains optimal and best values in many more instances than ABC and ABC_Nbr. To complement the numerical analysis, comparison graphs depicting (a) minimum *Stretch* values, (b) average *Stretch* values and (c) average time taken by these algorithms over 5 runs for the large Rectangular Grids are shown respectively in Fig. 4(a), (b) and (c). Similar illustration is given for large HB graphs in Fig. 5. The comparison clearly indicates the superiority of GVNS over ABC and ABC_Nbr.

The results of GVNS, ABC and ABC_Nbr are compared statistically on all the instances of Set A and B using two-way ANOVA without repetition test with 5% level of significance for S_T_{min}, S_T_{avg} and t_{avg}. It shows that there is a significant difference among the mean values of S_T_{min} over the three algorithms, whereas there is no significant difference among the mean values of S_T_{avg} of these algorithms. A Tukey's HSD test for pairwise comparison on S_T_{min} indicates that the three algorithms are significantly different from each other. Similar test done on the running times of the algorithms reveal that there is no significant difference between the mean values of t_{avg} of ABC_Nbr and ABC, while the mean values of t_{avg} are significantly different for the pairs (ABC, GVNS) and (ABC_Nbr, GVNS).

5 Conclusion

In this paper, a General Variable Neighborhood Search (GVNS) is proposed for MSSTP which uses the well known spanning tree algorithms for generating initial solutions. Six problem specific neighborhood techniques are designed which help in an exhaustive search of the solution space. Extensive experiments are conducted on various types of graphs in order to asses the performance of the proposed algorithm. Further, the results are compared with the adapted version of ABC (initially proposed for the tree t-spanner problem in the literature) and ABC_Nbr. Effectiveness of GVNS is clearly indicated through the results obtained by the three approaches in a majority of instances.

We learnt an interesting lesson from our testing that goes beyond the resolution of this specific problem. The variable neighborhood search template

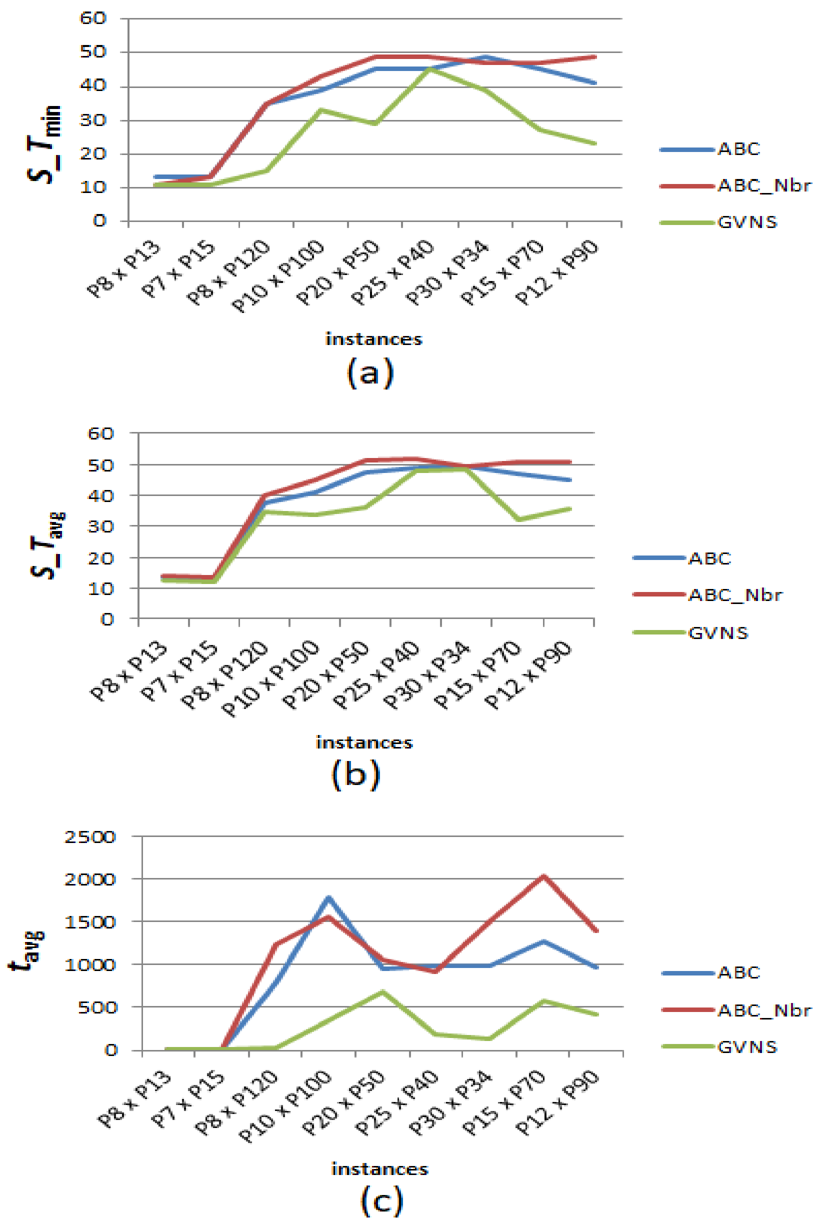

Fig. 4 Comparison of (a) minimum *Stretch* values, (b) average *Stretch* values and (c) average time taken by ABC, ABC_Nbr and GVNS over 5 runs for large Rectangular Grids.

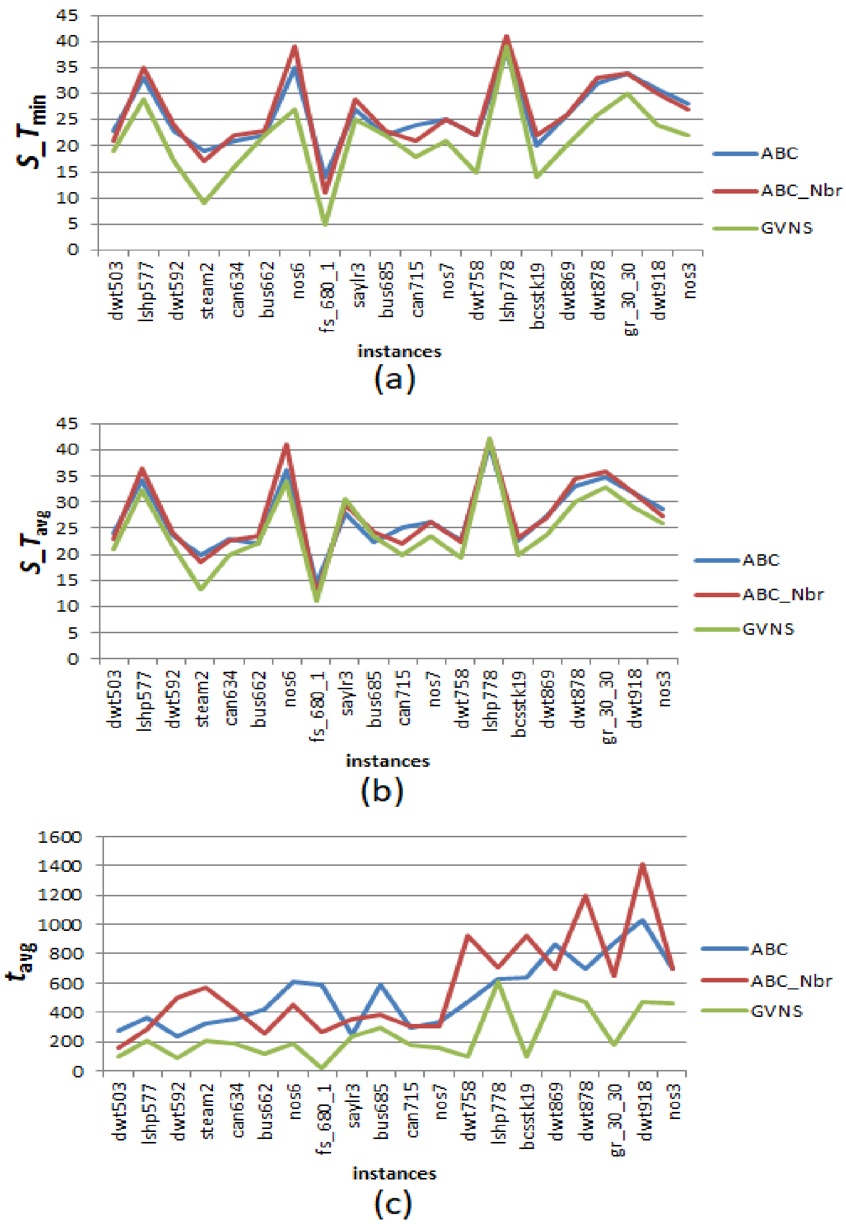

Fig. 5 Comparison of (a) minimum *Stretch* values, (b) average *Stretch* values and (c) average time taken by ABC, ABC_Nbr and GVNS over 5 runs for large HB graphs.

Table 4 Comparison of results obtained by ABC, ABC_Nbr and GVNS for Set A Graphs

	Graphs	W_n	K_n	S_n	$K_{n_1,n_2,\ldots,n_k}$	T_n	$P_m \times P_n$ (small)	$P_m \times P_n$ (large)	$TR_{m,n}$ (small)	$TR_{m,n}$ (large)
	size	[5, 1500]	[5, 1400]	[10, 1495]	[8, 1500]	[10, 1485]	[6, 100]	[104, 1080]	[12, 75]	[150, 1500]
	O_a	2.0	2.0	2.0	3.0	6.5	5.2	15.7	4.2	20.2
ABC	S_T_{min}	14.8	10.2	7.3	9.7	18.7	8.1	36.1	5.3	32.1
	S_T_{avg}	15.47	10.54	7.42	9.92	19.34	8.29	38.02	5.67	33.64
	t_{avg}	419.46	1365.04	422.62	711.63	361.59	5.77	866.71	3.43	998.81
	#opt	2	3	4	1	7	3	-	3	-
	#best	2	3	4	1	8	3	1	3	-
ABC_Nbr	S_T_{min}	10.8	7.1	4.9	6.7	18.9	7.9	38.1	5.2	33.7
	S_T_{avg}	11.58	7.34	5.17	6.94	19.66	8.38	40.73	5.42	35.47
	t_{avg}	393.65	1403.72	437.40	612.22	457.86	5.96	1083.09	3.70	1052.47
	#opt	6	6	9	7	6	3	-	4	-
	#best	6	6	9	7	9	3	1	4	-
GVNS	S_T_{min}	**2.0**	**2.1**	**2.0**	**3.0**	**17.2**	**5.9**	**25.9**	**4.4**	**27.3**
	S_T_{avg}	**5.49**	**4.24**	**3.39**	**4.32**	19.56	8.87	**32.46**	5.53	36.62
	t_{avg}	**0.26**	**72.00**	**42.57**	**39.07**	**165.90**	**1.31**	**265.44**	**0.58**	**524.31**
	#opt	**18**	**16**	**18**	**17**	**9**	**7**	-	**8**	-
	#best	**18**	**17**	**18**	**17**	**14**	**9**	**9**	**9**	**9**

Table 5 Comparison of results obtained by ABC, ABC_Nbr and GVNS for HB Graphs

	size	small ($\|N\| \leq 100$)	medium ($100 < \|N\| \leq 500$)	large ($500 < \|N\| \leq 1000$)
	# inst	15	35	20
ABC	S_T_{min}	9.2	17.1	26.0
	S_T_{avg}	9.36	17.81	27.05
	t_{avg}	5.62	84.08	526.22
	#best	7	3	3
ABC_Nbr	S_T_{min}	8.5	16.43	26.3
	S_T_{avg}	8.64	17.11	27.34
	t_{avg}	6.86	92.95	570.39
	#best	8	6	-
GVNS	S_T_{min}	**7.8**	**13.5**	**21.0**
	S_T_{avg}	**8.51**	**15.94**	**24.79**
	t_{avg}	**2.12**	**46.84**	**243.77**
	#best	**15**	**35**	**19**

clearly benefits from a relatively large number of neighbors. We invite researchers in others problems to test this point to advance the knowledge of the methodology.

Acknowledgment

The first author would like to thank University Grants Commission (UGC), India for providing RGNF fellowship [RGNF-2014-15-SC-UTT-74975] during the research. Rafael Martí is partially funded by the Spanish Ministry of Science and Innovation with grant ref. PGC2018-0953322-B-C21 MCIU/AEI/FEDER-UE.

Declarations

Funding

The funding details have been provided in Acknowledgement section.

Conflicts of interest

The authors have no conflicts of interest to declare that are relevant to the content of this article.

Availability of data and material

All data generated or analysed during this study are included in this article (See Appendix A).

Code availability

The source code will be made available on request.

Author's contributions

Not applicable

Ethics approval

Not applicable

Consent to participate

Not applicable

Consent for publication

All the authors agree for publication of this manuscript.

References

1. Wu, B.Y., Chao, K.M.: Spanning Trees and Optimization Problems, CHAPMAN & HALL/CRC, Washington, D.C. (2004)
2. Chen, G., Chen, S., Guo, W., Chen, H.: The multi-criteria minimum spanning tree problem based genetic algorithm, Inf. Sci., 177(22), 5050–5063 (2007). https://doi.org/10.1016/j.ins.2007.06.005
3. Ozeki, K., Yamashita, T.: Spanning trees: A survey, Graphs Combin., 27(1), 1–26 (2011). https://doi.org/10.1007/s00373-010-0973-2
4. Madkour, A., Aref, W.G., Rehman, F.U., Rahman, M.A., Basalamah, S.: A Survey of Shortest-Path Algorithms, arXiv preprint arXiv:1705.02044 (2017)
5. Brass, P., Vigan, I., Xu, N.: Shortest path planning for a tethered robot, Comput. Geom., 48(9), 732–742 (2015). https://doi.org/10.1016/j.comgeo.2015.06.004
6. Cintrano, C., Chicano, F., Alba, E.: Facing robustness as a multi-objective problem: A bi-objective shortest path problem in smart regions, Inf. Sci., 503, 255–273 (2019). https://doi.org/10.1016/j.ins.2019.07.014
7. Widmayer, P.: On shortest paths in VLSI design. Technical report, ACM Digital Library, Albert-Ludwigs University at Freiburg (1990)
8. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms, MIT Press, (2009).
9. Singh, K., Sundar, S.: Artifical bee colony algorithm using problem-specific neighborhood strategies for the tree t-spanner problem, Appl. Soft Comput., 62, 110–118 (2018). https://doi.org/10.1016/j.asoc.2017.10.022
10. Peleg, D., Upfal, E.: A trade-off between space and efficiency for routing tables, J. ACM, 36(3), 510–530 (1989). https://doi.org/10.1145/65950.65953
11. Hansen, P., Mladenović, N., Todosijević, R., Hanafi, S.: Variable neighborhood search: basics and variants. EURO J. Comput. Opt., 5(3), 423-454 (2017). https://doi.org/10.1007/s13675-016-0075-x.
12. Mladenović, N., Hansen, P.: Variable neighborhood search, Comput. Oper. Res., 24(11), 1097–1100 (1997). https://doi.org/10.1016/S0305-0548(97)00031-2
13. Sánchez-Oro, J., Pantrigo, J.J., Duarte, A.: Combining intensification and diversification strategies in VNS. An application to the Vertex Separation problem, Comput. Oper. Res., 52, 209–219 (2014). https://doi.org/10.1016/j.cor.2013.11.008
14. Sánchez-Oro, J., Gavara, A.M., Laguna, M., Martí, R., Duarte, A.: Variable Neighborhood Scatter Search for the Incremental Graph Drawing Problem, Comput. Optim. Appl., 68(3), 775–797 (2017). https://doi.org/10.1007/s10589-017-9926-5
15. Rodríguez-García, M.A., Sánchez-Oro, J., Rodriguez-Tello, E., Monfroy, E., Duarte, A.: Two-dimensional bandwidth minimization problem: Exact and heuristic approaches, Knowl. Based Syst., 214, 106–651 (2021). https://doi.org/10.1016/j.knosys.2020.106651
16. Peleg, D., Ullman, J.D.: An optimal synchronizer for the hypercube, SIAM J. Comput., 18(4), 740–747 (1989). https://doi.org/10.1137/0218050

17. Liebchen, C., Wünsch, G.: The zoo of tree spanner problems, Discret. Appl. Math., 156(5), 569–587 (2008). https://doi.org/10.1016/j.dam.2007.07.001
18. Cai, L., Corneil, D.G.: Tree spanners, SIAM J. Discret. Math., 8(3), 359–387 (1995). https://doi.org/10.1137/S0895480192237403
19. Álvarez-Miranda, E., Sinnl, M.: Mixed-integer programming approaches for the tree t-spanner problem, Optim. Lett., 13(7), 1693–1709 (2019). https://doi.org/10.1007/s11590-018-1340-0
20. Cheong, O., Haverkort, H., Lee, M.: Computing a minimum-dilation spanning tree is NP-hard, Comput. Geom., 41(3), 188–205 (2008). https://doi.org/10.1016/j.comgeo.2007.12.001
21. Gaiowski, M.F.A.M., Souza, C.C.: Minimum dilation geometric spanning trees, In: Proceedings of XLIII Simposio Brasileiro de Pesquisa Operacional (SBPO), pp. 1824–1835. Ubatuba-SP, Brazil (2011)
22. Brandt, A.F., Gaiowski, M.F.A.M., Rezende, P.J., Souza, C.C.: Computing Minimum Dilation Spanning Trees in Geometric Graphs, In: 21st International Conference on Computing and Combinatorics (COCOON), pp. 297–309. Springer, Beijing, China (2015)
23. Lin, L., Lin, Y.: The minimum stretch spanning tree problem for typical graphs, arXiv preprint arXiv:1712.03497 (2017)
24. Lin, L., Lin, Y.: Optimality computation of the minimum stretch spanning tree problem, Appl. Math. Comput., 386, 125502 (2020). https://doi.org/10.1016/j.amc.2020.125502
25. Lin, L., Lin, Y.: The Minimum Stretch Spanning Tree Problem for Hamming Graphs and Higher-Dimensional Grids, J. Interconnect. Netw., 20(1), 2050004:1–2050004:15 (2020). https://doi.org/10.1142/S0219265920500048
26. Boksberger, P., Kuhn, F., Wattenhofer, R.: On the approximation of the minimum maximum stretch tree problem. Technical report 409, Department of Computer Science, ETH Zurich (2003)
27. Kardam, Y.S., Srivastava, K.: General Variable Neighborhood Search for the Minimum Stretch Spanning Tree Problem, In: Computational Methods and Data Engineering (ICMDE), pp. 149–164. Springer, Sonipat, Delhi-NCR, India (2021). https://doi.org/10.1007/978-981-15-6876-3_12
28. Hansen, P., Mladenović, N.: Variable Neighborhood Search, R. Martí, P. Pardalos, M. Resende (eds) Handbook of Heuristics, 759–787 (2018)
29. Duarte, A., Oro, J.S., Mladenović, N., Todosijević, R.: Variable Neighborhood Descent, R. Martí, P. Pardalos, M. Resende (eds) Handbook of Heuristics, 341–367 (2018)
30. Martí, R., Resende, M., Pardalos, P.: Handbook of Heuristics, Springer International Publishing (2018)
31. West, D.B.: Introduction to graph theory, Prentice Hall (2001)

A Appendix

Note- In all the tables the results of instances for which the performance of GVNS is same as that of ABC or ABC_Nbr are shown in bold whereas the values in bold with asterisk show the improvement of our algorithm over ABC and ABC_Nbr.

Table A1 Comparison of results obtained by ABC, ABC_Nbr and GVNS for Wheel Graphs

Graphs	**size**	**Opt**	**ABC**			**ABC_Nbr**			**GVNS**		
			S_T_{min}	S_T_{avg}	t_{avg}	S_T_{min}	S_T_{avg}	t_{avg}	S_T_{min}	S_T_{avg}	t_{avg}
W_5	5	2	2	2.6	0.95	2	2.4	1.16	**2**	2.8	0.00
W_7	7	2	2	2.6	2.19	2	2.0	1.90	**2**	2.0	0.01
W_{10}	10	2	3	3.0	2.14	2	2.0	1.94	**2**	2.0	0.01
W_{15}	15	2	3	3.0	2.71	2	2.0	2.01	**2**	2.0	0.01
W_{20}	20	2	3	3.6	3.29	2	2.0	3.06	**2**	2.0	0.01
W_{30}	30	2	4	4.0	5.31	2	2.0	3.09	**2**	2.4	0.02
W_{50}	50	2	5	5.8	3.42	3	3.2	3.34	**2***	2.4	0.02
W_{70}	70	2	6	6.8	4.61	4	4.2	4.65	**2***	6.4	0.01
W_{100}	100	2	8	8.2	10.53	3	4.6	10.35	**2***	4.2	0.06
W_{150}	150	2	9	9.8	29.50	5	6.0	27.16	**2***	6.6	0.07
W_{200}	200	2	11	12.0	24.02	7	8.0	45.31	**2***	2.0	0.12
W_{500}	500	2	19	20.0	228.40	16	16.3	220.24	**2***	6.7	0.33
W_{700}	700	2	24	25.3	288.25	17	19.7	365.20	**2***	8.3	0.46
W_{900}	900	2	28	28.7	815.44	20	20.0	759.71	**2***	6.3	0.68
W_{1080}	1080	2	31	32.7	778.47	23	26.0	629.06	**2***	7.3	0.80
W_{1085}	1085	2	32	32.7	1465.54	26	26.3	1390.11	**2***	16.7	0.58
W_{1495}	1495	2	38	38.7	1728.05	30	31.0	1601.33	**2***	13.7	0.75
W_{1500}	1500	2	38	39.0	2157.40	28	30.7	2016.04	**2***	5.0	0.67

Table A2 Comparison of results obtained by ABC, ABC_Nbr and GVNS for Complete Graphs

Graphs	**size**	**Opt**	**ABC**			**ABC_Nbr**			**GVNS**		
			S_T_{min}	S_T_{avg}	t_{avg}	S_T_{min}	S_T_{avg}	t_{avg}	S_T_{min}	S_T_{avg}	t_{avg}
K_5	5	2	2	2.0	2.79	2	2.0	1.85	**2**	2.0	0.00
K_7	7	2	2	2.0	3.84	2	2.0	1.90	**2**	2.0	0.00
K_9	9	2	2	2.8	4.12	2	2.0	1.95	**2**	2.0	0.02
K_{10}	10	2	3	3.4	2.37	2	2.0	1.98	**2**	2.4	0.01
K_{15}	15	2	4	4.0	2.41	2	2.0	2.13	**2**	2.4	0.08
K_{20}	20	2	4	4.4	4.69	2	2.2	2.81	**2**	2.8	0.01
K_{25}	25	2	5	5.4	3.23	3	3.0	3.22	**2***	2.6	0.11
K_{30}	30	2	6	6.0	5.53	3	3.6	3.61	**2***	3.8	1.58
K_{50}	50	2	7	7.8	5.83	4	4.0	5.35	**2***	4.3	0.05
K_{100}	100	2	10	10.0	27.42	6	6.2	19.87	**4***	5.6	1.67
K_{500}	500	2	16	16.0	550.23	11	11.7	508.67	**2***	6.7	10.28
K_{600}	600	2	16	17.0	1172.04	12	12.0	1017.67	**2***	5.2	12.06
K_{900}	900	2	18	18.0	2213.86	14	14.0	1912.42	**2***	8.0	58.33
K_{1000}	1000	2	19	19.0	3034.78	13	13.7	2735.63	**2***	4.7	72.53
K_{1080}	1080	2	19	19.7	3023.04	14	14.3	2709.95	**2***	5.3	313.82
K_{1300}	1300	2	19	20.3	6142.52	15	15.0	7237.55	**2***	6.3	350.25
K_{1400}	1400	2	21	21.3	7007.04	14	15.0	7696.68	**2***	6.0	403.17

Table A3 Comparison of results obtained by ABC, ABC_Nbr and GVNS for Split Graphs

Graphs	size	Opt	ABC			ABC_Nbr			GVNS		
			S_T_{min}	S_T_{avg}	t_{avg}	S_T_{min}	S_T_{avg}	t_{avg}	S_T_{min}	S_T_{avg}	t_{avg}
S_{10}	10	2	2	2.0	2.49	2	2.0	1.99	**2**	2.0	0.00
S_{12}	12	2	2	2.0	2.64	2	2.0	0.81	**2**	2.0	0.00
S_{12}	12	2	2	2.0	2.28	2	2.0	2.02	**2**	2.0	0.01
S_{15}	15	2	2	2.0	1.43	2	2.0	1.70	**2**	2.0	0.00
S_{15}	15	2	3	3.0	2.27	2	2.0	2.06	**2**	2.0	0.03
S_{20}	20	2	3	3.0	2.35	2	2.0	2.21	**2**	2.2	0.01
S_{35}	35	2	4	4.0	2.70	2	2.0	2.81	**2**	2.8	0.02
S_{35}	35	2	4	4.0	2.84	2	2.4	6.07	**2**	2.8	0.04
S_{50}	50	2	4	4.0	3.37	2	2.4	5.10	**2**	2.8	0.18
S_{50}	50	2	5	5.4	5.09	3	3.6	3.76	**2***	3.4	1.66
S_{100}	100	2	6	6.3	8.21	4	4.0	8.71	**2***	2.5	2.31
S_{200}	200	2	12	12.0	80.87	8	8.0	83.58	**2***	3.5	10.03
S_{500}	500	2	8	8.0	181.87	5	5.7	135.09	**2***	2.3	23.82
S_{600}	600	2	13	13.3	378.73	9	9.3	488.85	**2***	3.4	45.23
S_{800}	800	2	12	12.0	396.44	7	7.7	400.84	**2***	4.6	43.01
S_{1085}	1085	2	19	19.3	2325.29	14	14.0	2208.96	**2***	5.3	123.22
S_{1200}	1200	2	16	16.3	2370.40	11	11.3	2158.52	**2***	6.7	213.57
S_{1495}	1495	2	15	15.0	1837.80	10	10.7	2360.03	**2***	8.7	303.04

Table A4 Comparison of results obtained by ABC, ABC_Nbr and GVNS for Complete k-Partite Graphs

Graphs	size	Opt	ABC			ABC_Nbr			GVNS		
			S_T_{min}	S_T_{avg}	t_{avg}	S_T_{min}	S_T_{avg}	t_{avg}	S_T_{min}	S_T_{avg}	t_{avg}
$K_{3,2,3}$	8	3	3	3.0	1.74	3	3.0	1.94	**3**	3.0	0.00
$K_{5,3,4,6}$	18	3	4	4.0	3.63	3	3.0	2.33	**3**	3.0	0.10
$K_{2,2,2,2,2,2,2,2,2,2}$	20	3	4	4.2	3.92	3	3.0	2.37	**3**	3.2	0.04
$K_{7,5,9,2}$	23	3	5	5.2	3.26	3	3.2	2.56	**3**	3.2	0.06
$K_{2,3,7,4,9}$	25	3	5	5.2	2.88	3	3.6	3.80	**3**	3.2	0.67
$K_{5,10,15}$	30	3	5	5.2	6.30	3	3.6	3.04	**3**	3.8	1.03
$K_{3,3,3,3,3,3,3,3,3,3}$	30	3	6	5.8	4.00	3	3.8	3.73	**3**	3.8	0.02
$K_{5,5,5,5,5,5,5}$	35	3	6	6.2	7.16	4	4.0	3.28	**3***	4.0	0.03
$K_{7,7,7,7,7,7,7}$	49	3	7	7.0	4.99	4	4.0	7.52	**3***	4.2	0.14
$K_{10,10,10,10,10}$	50	3	7	7.4	9.21	4	4.4	7.91	**3***	4.0	0.33
$K_{50,50}$	100	3	9	10.3	19.02	7	7.0	12.42	**3***	4.5	2.56
$K_{80,20,100}$	200	3	12	12.7	58.18	8	8.0	46.45	**3***	4.7	7.13
$K_{100,200,150,50}$	500	3	16	16.0	527.62	10	11.0	419.01	**3***	5.6	12.22
$K_{300,200,100}$	600	3	17	17.0	646.69	12	12.3	676.04	**3***	4.7	40.17
$K_{200,200,200,200}$	800	3	18	18.0	1699.33	14	14.0	1114.53	**3***	5.5	61.45
$K_{150,100,200,400,235}$	1085	3	19	19.7	3043.76	14	14.0	2806.16	**3***	6.4	223.30
$K_{500,500,500}$	1500	3	21	21.7	6056.08	16	16.0	5294.64	**3***	6.6	315.00

Table A5 Comparison of results obtained by ABC, ABC_Nbr and GVNS for Triangular Grids

Graphs	size	Opt	ABC			ABC_Nbr			GVNS		
			S_T_{min}	S_T_{avg}	t_{avg}	S_T_{min}	S_T_{avg}	t_{avg}	S_T_{min}	S_T_{avg}	t_{avg}
T_3	10	3	3	3.0	2.28	3	3.0	1.94	**3**	3.0	0.01
T_4	15	4	4	4.0	2.18	4	4.0	2.05	**4**	4.0	0.01
T_5	21	5	5	5.0	2.26	5	5.0	2.22	**5**	5.0	0.01
T_6	28	5	5	5.8	2.44	5	5.0	3.20	**5**	5.0	0.36
T_7	36	6	6	6.4	2.87	6	6.2	4.99	**6**	6.4	0.16
T_8	45	7	7	7.4	3.23	7	7.2	5.03	**7**	7.2	1.04
T_9	55	7	7	8.2	4.54	8	8.4	5.20	**7**	8.4	2.88
T_{10}	66	8	9	9.2	4.86	9	9.4	8.42	**8***	9.2	0.53
T_{11}	78	9	10	10.4	7.74	10	10.4	10.35	**9***	10.4	3.01
T_{15}	136	11	14	14.0	11.78	13	14.0	24.11	**13**	15.2	8.93
T_{20}	231	15	18	18.7	26.59	17	18.3	62.91	19	19.3	2.91
T_{25}	351	18	24	24.7	79.23	24	24.7	95.16	**19***	22.0	69.82
T_{30}	496	21	29	30.3	148.51	27	28.7	222.77	**23***	29.0	52.37
T_{40}	861	28	35	36.3	635.82	37	38.0	779.54	40	44.7	550.32
T_{45}	1081	31	44	44.3	1188.06	43	44.3	1966.39	45	46.7	1093.54
T_{50}	1326	35	47	48.7	1871.75	49	50.7	1473.60	**37***	46.0	486.92
T_{53}	1485	37	51	52.3	2152.85	55	57.0	3115.78	**43***	51.0	547.42

Table A6 Comparison of results obtained by ABC, ABC_Nbr and GVNS for Rectangular Grids

Graphs	size	Opt	ABC			ABC_Nbr			GVNS		
			S_T_{min}	S_T_{avg}	t_{avg}	S_T_{min}	S_T_{avg}	t_{avg}	S_T_{min}	S_T_{avg}	t_{avg}
$P_2 \times P_3$	6	3	3	3.0	1.47	3	3.0	1.15	**3**	3.0	0.00
$P_2 \times P_5$	10	3	3	3.0	2.43	3	3.0	1.57	**3**	3.0	0.00
$P_2 \times P_{10}$	20	3	3	3.8	2.45	3	3.0	2.19	**3**	3.0	0.13
$P_5 \times P_{10}$	50	5	9	9.0	3.18	7	8.6	4.89	**5***	9.0	2.38
$P_9 \times P_{11}$	99	9	13	13.0	8.15	13	13.0	9.01	**11***	15.4	3.22
$P_2 \times P_{50}$	100	3	7	7.0	8.90	7	7.0	7.39	**3***	7.8	0.21
$P_4 \times P_{25}$	100	5	11	11.0	8.41	11	11.0	8.54	**5***	11.8	0.69
$P_5 \times P_{20}$	100	5	11	11.4	9.32	11	12.2	7.68	**9***	13.0	2.13
$P_{10} \times P_{10}$	100	11	13	13.4	7.60	13	14.6	11.18	**11***	13.8	3.02
$P_8 \times P_{13}$	104	9	13	13.4	11.67	11	13.8	8.14	**11**	12.3	5.62
$P_7 \times P_{15}$	105	7	13	13.0	7.30	13	13.4	8.58	**11***	11.8	9.24
$P_8 \times P_{120}$	960	9	35	37.4	797.00	35	40.2	1246.35	**15***	34.6	30.32
$P_{10} \times P_{100}$	1000	11	39	41.0	1792.54	43	45.0	1566.63	**33***	33.6	340.31
$P_{20} \times P_{50}$	1000	21	45	47.4	946.43	49	51.4	1067.57	**29***	36.2	692.67
$P_{25} \times P_{40}$	1000	25	45	48.6	995.80	49	51.8	910.49	**45**	47.8	180.83
$P_{30} \times P_{34}$	1020	31	49	49.4	993.90	47	49.4	1501.77	**39***	48.2	125.96
$P_{15} \times P_{70}$	1050	15	45	47.0	1280.35	47	50.6	2043.70	**27***	32.2	577.65
$P_{12} \times P_{90}$	1080	13	41	45.0	975.40	49	51.0	1394.62	**23***	35.4	426.34

Table A7 Comparison of results obtained by ABC, ABC_Nbr and GVNS for Triangulated Rectangular Grids

Graphs	**size**	**Opt**	**ABC**			**ABC_Nbr**			**GVNS**		
			S_T_{min}	S_T_{avg}	t_{avg}	S_T_{min}	S_T_{avg}	t_{avg}	S_T_{min}	S_T_{avg}	t_{avg}
$TR_{3,4}$	12	3	3	3.0	2.40	3	3.0	2.01	**3**	3.0	0.01
$TR_{4,4}$	16	4	4	4.0	2.20	4	4.0	2.11	**4**	4.0	0.02
$TR_{4,5}$	20	4	5	5.0	2.25	4	4.4	3.88	**4**	4.2	0.36
$TR_{4,6}$	24	4	5	5.0	2.38	5	5.0	2.31	**4***	5.0	0.01
$TR_{5,5}$	25	5	5	5.4	2.41	5	5.0	2.37	**5**	5.0	0.05
$TR_{5,7}$	35	5	6	6.4	3.31	6	6.2	4.00	**5***	6.6	0.07
$TR_{3,15}$	45	3	5	5.8	3.86	5	5.2	4.00	**3***	6.0	0.02
$TR_{5,10}$	50	5	7	7.4	3.26	7	7.2	4.79	**5***	7.2	0.23
$TR_{5,15}$	75	5	8	9.0	8.83	8	8.8	7.80	**7***	8.8	4.47
$TR_{10,15}$	150	10	15	15.4	14.74	14	15.2	28.06	**13***	15.2	17.61
$TR_{11,15}$	165	11	14	15.4	27.24	16	16.4	27.40	**13***	15.6	12.41
$TR_{20,25}$	500	20	28	28.8	199.20	28	30.0	175.28	**24***	31.4	144.72
$TR_{15,40}$	600	15	29	30.4	419.92	31	31.6	455.46	**26***	31.0	161.43
$TR_{20,30}$	600	20	31	31.8	365.54	30	32.6	647.88	**26***	33.8	148.37
$TR_{8,120}$	960	8	29	31.2	1071.31	32	33.6	1162.95	**15***	32.6	552.68
$TR_{33,40}$	1320	33	46	48.6	1468.93	48	50.4	1922.50	**41***	54.2	978.90
$TR_{35,40}$	1400	35	49	50.2	2662.85	51	53.2	1788.92	**43***	61.4	1526.80
$TR_{30,50}$	1500	30	48	51.0	2759.56	53	56.2	3263.75	**45***	54.4	1175.83

Table A8 Comparison of results obtained by ABC, ABC_Nbr and GVNS for small HB graphs

Graphs	**size**	**LB**	**ABC**			**ABC_Nbr**			**GVNS**		
			S_T_{min}	S_T_{avg}	t_{avg}	S_T_{min}	S_T_{avg}	t_{avg}	S_T_{min}	S_T_{avg}	t_{avg}
can24	24	2	5	5.0	2.34	4	4.0	4.69	**4**	4.0	0.12
bcspwr01	39	12	12	12.0	1.63	12	12.0	1.74	**12**	12.0	0.01
bcsstk01	48	3	8	8.0	3.27	7	7.2	4.07	**6***	6.8	1.01
bcspwr02	49	7	7	7.0	3.35	7	7.0	2.65	**7**	7.0	0.03
dwt59	59	6	10	10.0	3.65	10	10.0	4.04	**10**	10.0	1.97
can61	61	2	7	7.4	4.26	5	5.2	5.35	**3***	3.8	0.21
can62	62	9	9	9.0	4.00	9	9.0	4.13	**9**	9.0	0.05
dwt66	66	2	4	4.6	5.04	4	4.0	7.29	**4**	4.8	2.73
bcsstk02	66	2	8	8.2	15.59	4	4.8	15.98	**2***	4.8	6.92
dwt72	72	15	15	15.0	3.75	15	15.0	4.12	**15**	15.0	0.09
can73	73	4	9	9.6	5.45	9	9.4	9.08	**9**	9.2	1.28
ash85	85	2	10	10.2	9.48	10	10.0	12.85	**8***	9.6	2.31
dwt87	87	3	8	8.0	5.54	7	7.4	7.93	**6***	7.0	2.73
can96	96	2	16	16.0	6.28	16	16.0	7.69	**15***	15.6	6.73
nos4	100	3	10	10.4	10.65	8	8.6	11.24	**7***	9.0	5.66

Table A9 Comparison of results obtained by ABC, ABC_Nbr and GVNS for medium HB graphs

Graphs	**size**	**LB**	**ABC**			**ABC_Nbr**			**GVNS**		
			S_T_{min}	S_T_{avg}	t_{avg}	S_T_{min}	S_T_{avg}	t_{avg}	S_T_{min}	S_T_{avg}	t_{avg}
bcspwr03	118	9	9	9.2	8.57	9	9.2	17.67	**9**	9.2	6.14
bcsstk04	132	3	11	11.0	23.26	8	8.2	35.40	**5***	6.6	3.68
can144	144	2	25	25.0	18.20	24	24.0	14.60	**24**	24.0	2.16
lund_a	147	2	11	11.6	25.11	10	10.2	24.72	**8***	9.4	11.29
lund_b	147	2	11	11.8	15.47	10	10.8	17.10	**8***	9.0	7.74
bcsstk05	153	3	11	11.6	16.90	10	10.0	21.91	**7***	7.8	1.74
can161	161	2	16	16.8	27.42	17	17.0	16.86	**15***	16.4	13.60
dwt162	162	2	29	29.0	15.17	29	29.0	17.39	**29**	29.0	9.05
can187	187	2	31	31.0	20.02	30	30.8	27.92	**30**	30.6	33.39
dwt193	193	2	12	12.6	40.47	10	10.6	33.53	**6***	7.2	15.32
dwt209	209	5	14	14.4	27.32	14	14.2	32.22	**11***	12.4	22.97
dwt221	221	2	16	16.6	36.45	16	16.0	87.40	**13***	14.6	29.14
can229	229	5	18	19.0	39.66	18	19.2	47.21	**16***	18.8	35.18
steam1	240	2	14	14.2	29.58	13	13.0	61.88	**7***	11.6	32.47
dwt245	245	10	14	14.0	27.80	12	12.6	81.32	**10***	12.2	16.81
can256	256	2	14	14.2	65.79	11	11.0	48.83	**6***	9.2	6.35
lshp265	265	2	21	22.4	63.44	21	23.0	108.33	**19***	23.0	68.19
can268	268	2	15	15.8	61.31	11	12.0	84.00	**7***	9.6	13.29
bcspwr04	274	9	11	11.4	46.87	11	11.2	44.34	**10***	10.6	7.71
can292	292	4	14	14.2	60.22	13	13.2	78.09	**10***	12.0	128.79
ash292	292	2	16	17.8	80.46	17	17.6	61.27	**14***	15.2	50.03
dwt307	307	2	24	25.6	57.35	25	25.4	81.34	**23***	26.0	92.39
dwt310	310	2	16	17.4	110.80	16	17.2	93.12	**11***	15.8	42.21
dwt361	361	2	21	21.2	179.63	21	21.8	140.46	**15***	20.0	55.89
plat362	362	2	17	17.2	79.67	15	15.6	133.41	**10***	14.4	64.22
lshp406	406	2	27	28.8	166.76	29	30.2	234.06	**26***	30.4	117.82
dwt419	419	3	22	23.8	239.72	23	24.6	188.96	**21***	22.6	60.60
bcsstk06	420	3	15	16.6	89.99	15	15.6	191.73	**9***	13.2	37.34
bcsstk07	420	3	16	16.6	161.06	15	15.8	181.21	**10***	13.2	60.63
bcsstm07	420	3	17	17.4	144.43	14	15.6	152.38	**10***	13.2	25.82
bcspwr05	443	11	15	15.4	176.98	14	15.6	155.10	**14**	16.2	52.31
can445	445	6	22	22.4	94.87	21	22.2	117.21	**15***	21.0	125.94
pores_3	456	3	21	22.4	387.93	19	22.0	266.84	**17***	19.2	74.01
nos5	468	4	16	16.8	164.19	16	16.4	182.85	**11***	15.6	177.71
bus494	494	18	18	18.0	139.99	18	18.2	172.68	**18**	18.8	137.47

Table A10 Comparison of results obtained by ABC, ABC_Nbr and GVNS for large HB graphs

Graphs	size	LB	ABC			ABC_Nbr			GVNS		
			S_T_{min}	S_T_{avg}	t_{avg}	S_T_{min}	S_T_{avg}	t_{avg}	S_T_{min}	S_T_{avg}	t_{avg}
dwt503	503	2	23	24.0	278.29	21	22.8	154.38	**19***	21.0	96.39
lshp577	577	2	33	34.2	365.73	35	36.4	279.19	**29***	32.4	206.65
dwt592	592	2	23	23.8	233.88	24	24.2	495.31	**17***	21.8	86.38
steam2	600	2	19	19.8	321.75	17	18.6	568.70	**9***	13.4	208.60
can634	634	9	21	23.0	352.50	22	22.6	420.70	**16***	20.0	185.77
bus662	662	22	22	22.2	419.03	23	23.4	251.06	**22**	22.0	115.00
nos6	675	3	35	36.2	610.08	39	41.0	454.56	**27***	33.8	181.16
fs_680_1	680	4	14	14.4	591.89	11	12.2	266.94	**5***	11.0	20.99
saylr3	681	3	27	27.8	244.69	29	29.4	351.72	**25***	30.6	235.42
bus685	685	21	22	22.4	585.63	23	24.2	378.67	**22**	23.4	290.15
can715	715	9	24	25.2	295.35	21	22.0	301.87	**18***	19.8	178.80
nos7	729	3	25	26.2	328.22	25	26.2	304.19	**21***	23.4	156.92
dwt758	758	2	22	22.6	473.58	22	22.4	917.91	**15***	19.2	93.78
lshp778	778	2	38	40.8	629.72	41	42.0	700.47	39	42.2	606.24
bcsstk19	817	10	20	22.6	639.99	22	23.2	925.46	**14***	19.9	95.46
dwt869	869	2	26	27.4	864.70	26	27.0	698.96	**20***	23.8	538.90
dwt878	878	2	32	33.2	690.76	33	34.6	1194.86	**26***	30.2	470.32
gr_30_30	900	2	34	34.8	870.98	34	35.8	642.80	**30***	32.8	174.39
dwt918	918	2	31	31.8	1031.18	30	31.6	1408.65	**24***	29.1	471.96
nos3	960	2	28	28.6	696.52	27	27.2	691.43	**22***	26.0	462.08