Biobjective Evolutionary and Heuristic Algorithms for Intersection of Geometric Graphs

Rajeev Kumar[*]
Dept. Computer Sc. & Engg.
Indian Institute of Technology
Kanpur, UP 208 016, India
raj@iitk.ac.in

P.K. Singh
Dept. Computer Sc. & Engg.
Indian Institute of Technology
Kharagpur, WB 721 302, India
pksingh@cse.iitkgp.ernet.in

Bhargab B. Bhattacharya
Adv. Comp. Microelectro. Unit
Indian Statistical Institute
Kolkata, WB 700 018, India
bhargab@isical.ac.in

ABSTRACT

Wire routing in a VLSI chip often requires minimization of wire-length as well as the number of intersections among multiple nets. Such an optimization problem is computationally hard for which no efficient algorithm or good heuristic is known to exist. Additionally, in a biobjective setting, the major challenge to solve a problem is to obtain *representative* diverse solutions across the (near-) Pareto-front.

In this work, we consider the problem of constructing spanning trees of two geometric graphs corresponding to two nets, each with multiple terminals, with a goal to minimize the total edge cost and the number of intersections among the edges of the two trees. We first design simple heuristics to obtain the extreme points in the solution space, which however, could not produce diverse solutions. Search algorithms based on evolutionary multiobjective optimization (EMO) are then proposed to obtain diverse solutions in the feasible solution space. Each element of this solution set is a tuple of two spanning trees corresponding to the given geometric graphs. Empirical evidence shows that the proposed evolutionary algorithms cover a larger range and are much superior to the heuristics.

Categories and Subject Descriptors

I.2.8 [**Artificial Intelligence**]: Problem Solving, Control Methods and Search—*Heuristic Methods*; J.6 [**Computer-Aided Engineering**]: Computer-Aided Design.

General Terms

Algorithm, Design, Experimentation.

[*]Author is on sabbatical from CSE Department, IIT Kharagpur. He acknowledges receiving travel support from Research I Foundation, CSE Department, IIT Kanpur and other financial support from a project grant funded by MHRD, Govt. of India.

Keywords

Evolutionary multiobjective optimization, genetic algorithm, combinatorial optimization, geometric graph, spanning tree, Steiner tree, Pareto front, CAD for VLSI.

1. INTRODUCTION

In physical design of VLSI circuits, the *routing* phase consists of laying out the wiring paths on metal layers that electrically interconnect all the terminals of each net. The routing task is usually divided into two phases *global* or *loose* and *local* or *detailed* routing. In global routing the exact geometric details are ignored and only the loose routes are determined. It determines the open regions through which each wire should run. Detailed routing completes the point-to-point wiring by specifying geometric information such as location and width of wires and their layer assignments. Most of the optimization problems related to routing are known to be NP-hard, therefore, approximation algorithms or heuristics are generally used to solve them [24, 27].

For high performance chips it is important to route in such a way that all the signals meet their timing constraints. One of the typical optimization goals is to minimize the required wire-length for each net. This may require determination of a minimum rectilinear Steiner tree (RST), which is a computationally hard problem [12, 13]. Hwang [13] has shown that the cost of a minimum RST is at most one-and-a-half times the cost of a minimum spanning tree (MST). Thus finding an MST often serves as an initial step in such heuristic algorithms. An illustration of a MST from a set of eight points in a 2-dim plane, and the corresponding minimum RST with three additional Steiner points is included in Figure 1.

For routing multiple nets on a metal layer, minimizing total wire-length of the spanning trees is not the only criterion for routing. Two geometrically crossing edges (belonging to two distinct nets) cannot be routed on a single metal layer preserving their embeddings, as they would cause an undesirable electrical short. To make use of another routing layer, each crossing among the tree edges will need vias so that the wires can change layers. Increase in the number of vias may however decrease the yield and introduce parasitic capacitances, which in turn, may affect the operating speed of the chip. Thus, one should also minimize the number of intersections among the tree edges that belong to different nets. In this paper, our design strategy, therefore, is *simultaneously* minimizing the wire-length as well as the number of intersecting points between two spanning trees.

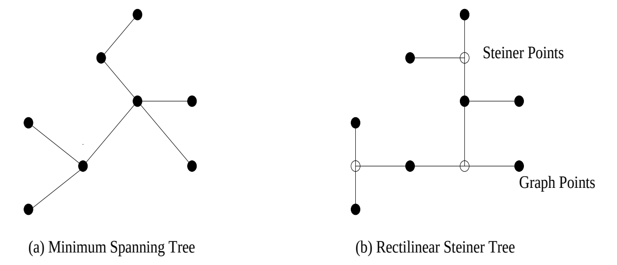

Figure 1: A MST and the corresponding Minimum RST with use of additional Stenier points.

Several theoretical results on minimizing the number of intersections among the edges of multiple spanning trees are well known. Tokunaga [28] addressed the problem of finding geometric spanning trees of bi-colored point sets on two simple geometric graphs[1] such that they intersect in as few points as possible. Bilo et al. [3] and Har-Peled and Indyk [11] presented greedy/approximation algorithms for crossing spanning trees. Recently, Kano et al. [17] solved a problem similar to that of Tokunaga but with a larger number of geometric graphs instead of two and suggested an upper bound on the number of intersections of the tree edges. Majumder et al. [22] studied a similar problem and reported heuristics for rectilinear Steiner trees of bi-colored point sets. Their heuristic first generates a geometric MST and then converts it to a rectilinear one. There are many other related studies too. For example, Abellanas et al. [1] and Akiyama and Urrutia [2] addressed the problem of embedding trees and alternating paths on bi-colored point sets, and Dumitrescu & Kaye [9] studied matching problem on colored point sets. For survey articles on related subjects, see Pach [25] and Kaneko & Kano [16]. There are many other spanning/Steiner problem instances having a bound on degree/diameter, capacitated trees or bounds for two parameters to be satisfied simultaneously, see Deo & Kumar [8], Marathe et al. [23]. Using evolutionary algorithms (EAs), Knowles & Corne [18] and Raidl & Julstrom [26] attempted to solve similar constrained minimum spanning tree problems.

We argue that use of heuristics may yield a *single* optimal solution, one in each objective, and may not yield many other equivalent solutions. With use of ϵ-constraint method, most other solutions obtained are located near the minimal region of the respective criterion of the Pareto-front, and thus do not form the complete Pareto-front.

However, the design problem considered in this paper is essentially multiobjective in nature. A multiobjective optimizer yields a set of all representative equivalent and diverse solutions; the set of all optimal solutions is the Pareto-front. In this work, we use evolutionary multiobjective optimizer (EMO) to obtain a (near-) optimal Pareto-front. However, black-box optimizers, e.g. EMOs, in solving such *hard* problems, have their own challenges and difficulties. For example, EMOs have been very effective for those problems for which one has a reference set of (optimal) solutions to assess quality of the obtained solution front [4, 6].

Since the problem is *hard* and the Pareto-front is *unknown*, the main issues in such problem instances are: how to assess the convergence, and how to obtain many representative diverse solutions across the Pareto-front. Most of the strategies to assess the convergence need a *reference* solution front which is not known for this problem. Moreover, most diversity preserving strategies attempt to find a *uniformly* distributed solution front, which may not be the case with an unknown problem.

In this work, we use two different multi-objective evolutionary algorithms (MOEAs) and get the solution-front. Then, in order to validate the solutions, we design two heuristics to obtain the extreme points nearer to their location. We observe that the solutions obtained by MOEAs for larger graphs are much inferior and do not cover the entire range. Therefore, we embed this knowledge in the solution evolving strategy, and obtain diverse solutions in the entire range. The solutions obtained by MOEAs are much superior to those obtained by the heuristics.

The rest of the paper is organized as follows. In Section 2, we describe the heuristics to obtain the solutions nearer to the (approximated) extreme points. In Sub-Section 3.1, we present a brief review of the issues and challenges in solving multiobjective real-world applications. We describe, in Sub-Section 3.2, the representation scheme for spanning tree and the genetic operators and evolutionary algorithm used. We include empirical results in Section 4 along with a comparison with different approaches. Finally, we draw conclusions in Section 5.

2. HEURISTIC ALGORITHMS

Problem Definition: Given two geometric graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ such that $V_1 \cap V_2 = \phi$ and $V_1 \cup V_2$ are in general position. We need to find the spanning trees T_1 and T_2 for G_1 and G_2, respectively, such that edges of the trees have as few intersections as possible with costs as less as possible. Thus, we have to simultaneously minimize both the objectives, namely, edge-cost and intersection points for the given geometric graphs.

First, we experiment with exact MSTs to get a representative solution for the minimum value of the total cost; such an MST may have a higher number of intersections.

Algorithm 1 Search over Minimal Cost Spanning Trees

```
1: Input: $G_1$ - graph 1, $G_2$ - graph 2
2: Output: a singleton set consisting of a tuple $(T_1, T_2)$ over
   MSTs from $G_1$ and $G_2$, respectively
3: Algorithm:
4: Initialize $PF$ as $\phi$
5: for all nodes $u_1$ of $G_1$ do
6:    Make $T_1$ considering $u_1$ as start node of the tree
7:    for all nodes $u_2$ of $G_2$ do
8:       Make $T_2$ considering $u_2$ as start node of the tree
9:       compute objective vector of tuple $(T_1, T_2)$
10:      Update the singleton set
11:   end for
12: end for
13: Output Point on the Pareto Front (PF)
```

[1] *Geometric graphs* $G = (V, E)$ are the graphs whose nodes (set V) are a collection of points on a plane in general positions, i.e., no three points are on a line, and their edges (set E) are a collection of straight line segments connecting a pair of nodes in set V. A graph G is said to be *simple* if two edges of G do not intersect except at their endpoints.

2.1 Finding Minimal Cost Spanning Trees

We compute MSTs T_1 and T_2 for graphs G_1 and G_2, respectively, using Prim's algorithm [5]. Since, Prim's algorithm may give different MSTs for different start nodes, we generate MSTs considering each node of the graph as the start node and compute the number of intersections for each MST. Thus, we do exhaustive search to look for minimal number of intersections for the minimum cost tree; the solution obtained thus is the optimal solution for the minimal cost. Algorithm 1 includes pseudocode for the exhaustive search over all the MSTs.

The solution thus obtained gives us one extreme solution, which cannot be optimized further. We obtain the best possible value for the first objective - the total cost. It is not possible to further improve upon this objective irrespective of the value of second objective - the number of intersections. Clearly, we have an associated value for the second objective also with this optimum value of first objective. Now, if we try to improve upon the value of second objective we would have to trade-off with the value of the cost. Hence, it is not possible to get any better solution than this with respect to the minimal cost. Also, there wouldn't be any other solution superior in the number of intersections for this minimal cost. This solution point is the extreme point in cost. In order to get the other trade-offs solutions, we device the heuristic given in Algorithm 2.

Algorithm 2 Heuristic for Fewer Intersection Points

```
1: Input: $G_1$ - graph 1, $G_2$ - graph 2
2: Output: a set of solution (Pareto-front (PF)) consisting
   of tuples $(T_1, T_2)$ from $G_1$ and $G_2$, respectively
3: Algorithm:
4: Initialize $PF$ as $\phi$
5: for all nodes $v_1$ of $G_1$ do
6:   Initialize $T_1 = \phi$, $C_1 = \phi$ and $U_1 = G_1$, where $U_1$
     is set of unconnected vertices and $C_2$ is set of tree
     (connected) vertices
7:   Delete $v_1$ from $U_1$ and append to $C_1$
8:   for all nodes $v_2$ of $G_2$ do
9:     Initialize $T_2 = \phi$, $C_2 = \phi$ and $U_2 = G_2$, where $U_2$
       is set of unconnected vertices and $C_2$ is set of tree
       (connected) vertices
10:    Delete $v_2$ from $U_2$ and append to $C_2$
11:    while $U_1 \neq \phi \vee U_2 \neq \phi$ do
12:      if $U_1 \neq \phi$ then
13:        Take $u'$ from $C_1$ and $v'$ from $U_1$ where number
           of intersections is least among $T_1 \cup T_2 \cup (u', v')$
14:        Append the edge $(u', v')$ to $T_1$
15:        Delete $v'$ from $U_1$ and append to $C_1$
16:      end if
17:      if $U_2 \neq \phi$ then
18:        Take $u''$ from $C_2$ and $v''$ from $U_2$ where number
           of intersections is least among $T_1 \cup T_2 \cup (u'', v'')$
19:        Append the edge $(u'', v'')$ to $T_2$
20:        Delete $v''$ from $U_2$ and append to $C_2$
21:      end if
22:    end while
23:    compute objective vector of tuple $(T_1, T_2)$
24:    Update $PF$
25:  end for
26: end for
27: Output $PF$
```

2.2 Heuristic for Fewer Intersections

Algorithm 2 produces solutions in the lower range of number of intersections. Within each iteration of the *while* loop it considers an edge (u, v) to append to the partial tree that minimizes the value of the second objective (number of intersections) keeping the value of first objective optimum. In each iteration, it selects an edge with least value of the first objective that does not increase the value of second objective; if it is not possible to select an edge that does not increase the value of second objective it considers an edge which increases the value of the first objective by least amount for least increase in value of the second objective.

This algorithm starts with a start node v_1 for T_1 from G_1 and a start node v_2 for T_2 from G_2. Further to make the trees, in each iteration, it chooses a least cost edge successively from G_1 and G_2 such that it is optimum for first objective and minimum for second objective.

We experiment with the other variants of the greedy criterion by changing the edge selection process. First, we experimented with selecting a random node from C (set of connected tree vertices) and finding a node from U (set of unconnected vertices) that makes it the best possible edge to append to the partial tree. Second, we experimented with changing roles of C and U, i.e., we selected a random node from U and find the best possible node from C. The rational behind these experiments is to convert this deterministic algorithm into a somewhat stochastic algorithm and find out (i) whether the complete greediness (the deterministic) is misleading, and (ii) whether it is possible to find a stochastic greedy heuristic that provides superior solutions. But none of these experiments yield good results. We found that selecting best possible edge deterministically yields better results compared to an edge whose endpoint is selected stochastically either from C or from U. We are currently investigating with other variants of the greedy criteria.

With these two heuristic algorithms we get two sets of extreme solutions; Algorithm 1 produces only a single solution at one extreme and Algorithm 2 produces a set of solutions at the other extreme. It not only provides best extreme solutions at either ends but also gives a fair idea of the range of solutions by which the Pareto-front is bounded. We'll show in Sub-Section 4.3 that this knowledge of the solution-space helps us to get superior quality solutions by the EMO.

3. EVOLUTIONARY OPTIMIZATION

3.1 Issues & Challenges

EAs have emerged as powerful black-box optimization tools to approximate solutions for NP-hard combinatorial optimization problems. In the multiobjective scenario, EAs often find effectively a set of mutually competitive solutions without needing much problem-specific information. For a wide-ranging review, a critical analysis of evolutionary approaches to multiobjective optimization (MOO) and many implementations of multiobjective EAs, see [6], and for various applications, see [4]. However, achieving proper diversity in the solution-set while approaching convergence is a challenge in MOO, especially for unknown problems.

3.1.1 Achieving Diversity

Many techniques and operators have been proposed to achieve diversity. The commonly used techniques for preventing genetic drift and promoting diversity are: sharing, mating restrictions, density count (crowding), clustering and pre-selection operators. These approaches can be grouped into two classes: parameter-based sharing and parameter-less sharing. The niching/sharing techniques have been commonly employed to find a diverse set of solutions although such techniques work best when one has a priori knowledge of the solution. It is a common experience of many researchers that sharing can be beneficial, but can also prove surprisingly ineffective if parameters are not properly tuned. Also, it is the experience of almost all researchers that proper tuning of sharing parameters is necessary for effective performance. In recent years, much work has been done on parameter-less diversity preserving approaches. Most of the newer MOEAs, (e.g., NSGA-II [6] and SPEA2 [6]) have now dispensed away or new MOEAs (e.g., PCGA [20]) do not use explicit parameters for diversity preserving. (The revised NSGA-II and SPEA2 use parameter-less crowding and clustering, respectively.)

3.1.2 Monitoring Convergence

A common metric for convergence is the distance metric, which finds distance of the obtained solution front from the true Pareto front; this is trivial for known problems. Such a metric is based on a reference front. In real-world search problems, location of the actual Pareto-front, by definition, is unknown. A commonly practiced approach to determine the reference front for unknown problems is to extract the reference front from the best solutions obtained so far, and the reference is incrementally updated with every generation in iterative refinement based algorithms.

As to on-line convergence metric (e.g., [7]) evaluates convergence towards a reference set, and is akin to monitoring hyper-volume measure; this may not be used effectively for unknown problems. Kumar and Rockett [20] proposed use of rank-histograms to assess movement of solution-front towards convergence.

3.1.3 Avoiding Local Convergence

For solving unknown problems there is a common concern whether or not the obtained solution set is close to the true Pareto-front. Apparently, it seemed that the EA had converged to the Pareto front but conceivably it may have got *stuck* at some sub-optimal point. Such a local minima cannot be detected for unknown problems, by most of the known metrics because a local front obtained may give excellent numerical values for both diversity and convergence [21].

3.1.4 Performance Assessment

Many metrics have been proposed for quantitative evaluation of the quality of solutions. Essentially, these metrics are used for the following three characteristics:

- Extent: Coverage of the solutions across the front;
- Diversity : Sampling of the solutions across the front;
- Convergence: Distance of the obtained solution-front from the *reference* front.

Some of these metrics are only applicable where the solution is known. In the case of unknown nature, the metrics are sensitive to the choice of the reference front. Some EA researchers (e.g., Knowles and Corne [19], Zitzler et al. [30]) gave detailed critical review of these measures. They have also shown the sensitivity of some of the metrics with respect to the arbitrary choice of the reference point/front.

3.2 Algorithm & Operators

From the viewpoint of EMO, the optimization problem attempted in this paper is characterized by the following features:

- No *a priori* knowledge of the solution space is available.
- There exists no information regarding a reference front.
- No experimental result from any polynomial time *good* approximation algorithm is available.

There are many MOEAs and their implementations. Seeing the hardness of the problem, we select those MOEAs, which are steady-state algorithms and use archives that can be updated with the genetic evolutions. For this, among the many such algorithms, we select NSGA-II [6] and Pareto Converging Genetic Algorithm (PCGA) [20]. Both these algorithms do not need much of problem-dependent knowledge. PCGA has an added advantage that it monitors convergence for unknown solution space through rank histograms, and thus, avoids wastage of CPU time in unnecessarily scheduling a particular run of the algorithm to a pre-determined number of iterations. A brief pseudocode of the algorithm is given in Algorithm 3, see [20] for details of the algorithm.

Algorithm 3 Pareto Converging GA

```
1: Input: $N$ - size of initial population and GA parameters
2: Output: a set of (near-) optimal solutions
3: Algorithm:
4: Generate an initial population of size $N$
5: Compute individual's objective vector
6: Pareto-rank the population and generate rank-histogram
7: while $I$ntra-island rank-ratio histogram does not satisfy stopping criteria do
8:     Select two parents using roulette wheel selection scheme
9:     Perform crossover and mutation to generate two offspring
10:    Compute objective vectors of offspring
11:    Pareto-rank the population including offspring
12:    Remove the two least fit individuals (with tie resolution) to keep the size $N$
13:    Generate rank-ratio histogram
14: end while
15: One while-loop for $I$nter-island rank-histogram satisfying stopping criterion
16: Output set of solutions
```

Algorithm 3 is a steady-state algorithm and can be seen as an example of $(\mu + 2)$ – Evolutionary Strategy (ES) in terms of its selection mechanism. In this algorithm, individuals are compared against the total population set according to the Pareto-ranking scheme [10], and the population is selectively moved towards convergence by discarding the lowest ranked individuals in each evolution. In doing so, we require no parameters. A pair of mates is randomly chosen biased in the sizes of the roulette wheel segments and crossed-over and/or mutated to produce offspring. The offspring are inserted into the population set according to their ranks against the whole population and the lowest ranked two individuals are eliminated to restore the population size to N. For a meaningful comparison of two real numbers during ranking, the floating-point precision of the objective values is restricted to a few units of precision. The process is iterated until a convergence criterion based on Intra-island rank-ratio and Inter-island rank histogram is achieved [20].

Chromosome Encoding. The efficiency of the evolutionary search depends how a problem (in this case, a spanning tree) is represented in a chromosome and the reproduction operators work on the encoding. There are many encoding schemes to represent spanning trees – see [15] for a detailed review and comparison. Raidl & Julstrom [15] proposed spanning trees to be represented directly as sets of the edges and have shown locality, heritability and computational efficiency of the edge sets for evolutionary search. In this work, we use edge-set scheme for representing spanning trees to exploring the search space.

Initial Population. Instead of using MSTs as individual solutions in initial population we generate random trees. We start with a random start node and select a random edge from the eligible list of edges to complete the tree. This is done to un-bias the randomly generated population from the links found in a MST. A procedure to generate initial population is given as Algorithm 4.

Algorithm 4 Initial Population

```
 1: Input: $G$ - graph
 2: Output: $T$ - spanning tree
 3: Algorithm:
 4: Initialize $T = \phi, C = \phi$ and $U = G$, where $U$ is set of
    unconnected nodes and $C$ is set of tree nodes
 5: Select a node $w$ randomly from $U$, delete it from $U$ and
    append to $C$
 6: while tree is not complete do
 7:    Select a node $v$ randomly from $U$ and a node $u$ randomly from C
 8:    Append this edge $(u, v)$ to $T$, delete $v$ from $U$ and
       append to $C$
 9: end while
10: Output $T$
```

Genetic Operators. We select crossover operator to provide strong habitability such that the generated trees consist of the parental edges as far as possible. We use roulette wheel selection to select two parents. Crossover operator is a variant of the crossover scheme proposed in [26]. Without using any other information, we select an edge that is common in both parents to start with. In each iteration we choose an edge that is common to both parents and incident to either of the endpoints of start/recently added edge. If no common edge is available we chose an edge from either of the parents. We chose a random edge to complete the tree if it is still not complete. Algorithm to crossover two edge-sets is given in Algorithm 5.

The mutation operators used in this work are again the variants of the operators defined by Raidl & Julstrom to work for edge-set encoding [26]. They designed all the four mutation operators based on the diameter information. In

Algorithm 5 Crossover operator

```
 1: Input: Parent1 and Parent2
 2: Output: $T$ - Offspring
 3: Algorithm:
 4: Initialize $T = \phi$, $C = \phi$ and $U = G$, where $U$ is a set
    of unconnected nodes and $C$ a set of connected (tree)
    nodes
 5: Initialize set $A1$ consisting of edges appearing in both
    parents and $A2$ with edges appearing in only one parent
 6: Select an edge randomly from $A1$
 7: while tree is not complete do
 8:    Select an edge randomly from $A1$; if $A1$ is empty select
       from $A2$
 9:    If $A2$ is also empty, select an edge randomly joining a
       random node in $C$ and a random node in $U$
10:    Append the edge to $T$
11: end while
12: Output $T$
```

our case, we do not need diameter values, therefore, we adapted their mutation operators to work for our work. Edge-delete mutation deletes an edge randomly and rebuilds the tree as in crossover scheme and takes a new edge when necessary (Algorithm 6).

Algorithm 6 Edge Delete Mutation

```
 1: Input: $T$ - tree
 2: Output: $T'$ - tree
 3: Algorithm:
 4: Delete an edge randomly from $T$
 5: Build $T'$ as in crossover using edges from $T$ when possible and new edges when necessary
 6: Output $T'$
```

Algorithm 7 Greedy Edge Replace Mutation

```
 1: Input: $T$ - tree
 2: Output: $T'$ - tree
 3: Algorithm:
 4: Delete an edge randomly from $T$
    (it disconnects a subtree rooted at node $v$)
 5: Compute a set $S$ that consists the nodes of this subtree
    including $v$
 6: Reconnect the tree with lowest weight edge $(u, v)$, where
    $u$ is from $V - S$
 7: Output $T'$
```

Our greedy-edge-replace mutation is similar to the original operator except depth information. The greedy-edge-replace mutation also deletes an edge from the tree. Deleting an edge disconnects the tree in two subtrees. The two subtrees are reconnected with a lowest weight edge (Algorithm 7). Another subtree-optimize mutation operator is illustrated in Algorithm 8.

4. RESULTS & DISCUSSIONS

We tested generation of dual objective spanning tree using the heuristics and evolutionary algorithms and selected benchmark data taken from Beasley's OR library[2]. The

[2]http://mscmga.ms.ic.ac.uk/info.html

Algorithm 8 Subtree Optimize Mutation

```
1: Input: $T$ - tree
2: Output: $T'$ - tree
3: Algorithm:
4: Select a node $v$ randomly
5: Compute a set $S$ consisting nodes of the subtree including $v$
6: Optimize this subtree rooted at $v$ as in random greedy heuristic (RGH)
7: Output $T'$
```

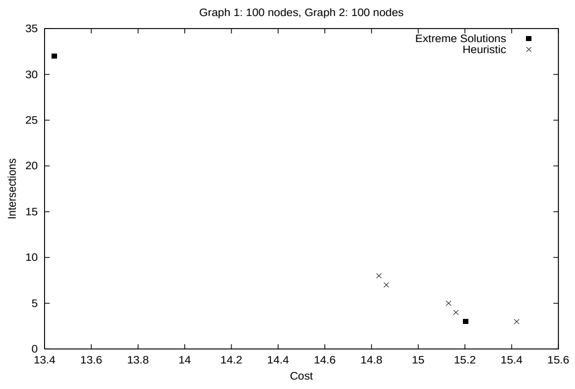

Figure 2: Set of solutions generated by heuristic Algorithms 1 and 2 for 100 node data. Extreme solutions for each of the objectives are also shown.

OR-Library is a collection of test data sets for a variety of Operations Research (OR) problems. We considered the Euclidean Steiner problem data of 50 and 100 nodes for this work.

4.1 Results from Heuristic Algorithms

First, we include, in Figure 2, results for 100 node data using both the heuristic Algorithms 1 and 2. As discussed in Section 2, we get a single solution for minimal cost region from Algorithm 1; this solution is the minimum edge-cost solution with the least number of intersections in comparison to other minimum cost solutions. From Algorithm 2, we obtain a set of solutions in the lower range of intersections. This heuristic is fine-tuned to yield solutions in the range of smaller number of intersections, thus we get one solution, which has the minimum number of intersection points; this solution is shown by a dark filled square in Figure 2. It can be seen that we are able to get solutions in the extreme range of each of the objectives, heuristics could not find a spread of solutions in the entire range of the objective values. Similar are the results for 50 node data.

4.2 Results from EMO

Next, we generated results from MOEAs for 50 and 100 node data. For this, first we generated random initial population, as discussed in Section 3.2. A representative set of randomly generated population is shown in Figure 3. Since, we have not generated initial population from the derivatives of MSTs, it can be seen from this figure that none of the points are located in the lower cost region. Though all the initial population is clustered in a narrow region, this is an indication of the hardness of the problem. For MOEA,

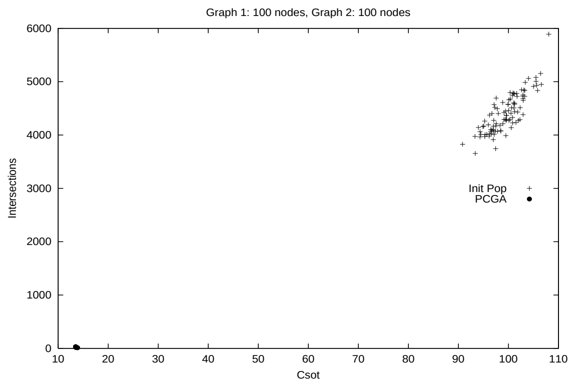

Figure 3: Plot showing randomly generated population for 100 node data.

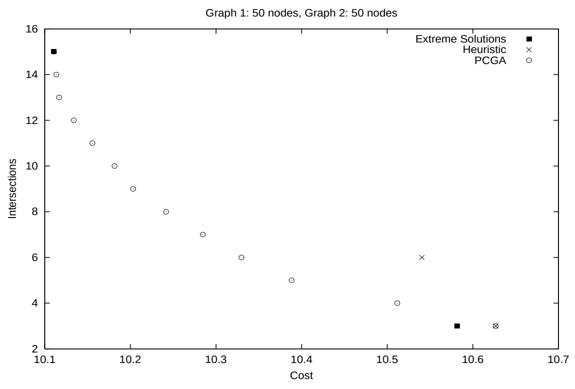

Figure 4: Solutions obtained form MOEA using a randomly initialized population from 50 node data. Extreme solutions are also shown.

we have taken population of 100 individuals, and probability of crossover and mutation is taken to be 0.8 and 1.0, respectively. We run the experiments for ten times each; in the rest of the section, we include on the average poorer results obtained by MOEA.

We include results obtained from MOEA using randomly initialized population for 50 and 100 node data in Figures 4 and 5. Few observations can be made from these two figures. Looking at solutions form 50 node data, one can see a good distribution of solutions between two extreme solutions. Thus, one can conclude that MOEA could effectively obtain good diverse solutions. Though nothing much can be said about the optimality of the obtained front, yet one can infer that both the extreme solutions have also been discovered by MOEA, indicating the fact that the obtained solution front is very close to the true Pareto-front.

However, this is not exactly the case with the solutions obtained by MOEA for 100 node data (Figure 5). On comparing this figure with the randomly generated population in Figure 3, MOEA has done a great job and the points have moved a lot during population evolution. The solutions obtained in low-cost, higher-intersection region are well distributed. However, there does not exist any solution in the other extreme of the solution space, i.e., the high-cost, lower-intersection region. The plot included for this result is the poorest among all the ten results obtained for this

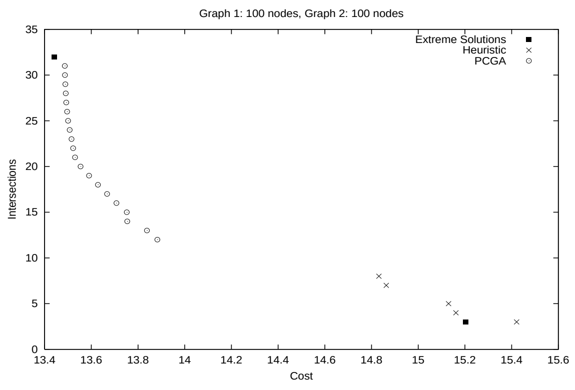

Figure 5: Solutions obtained form MOEA using a randomly initialized population from 100 node data. Extreme solutions are also shown.

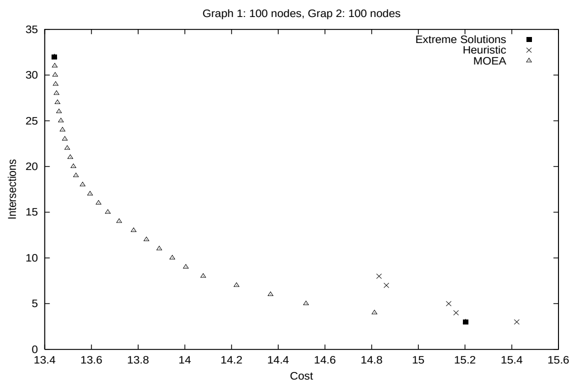

Figure 6: Solutions obtained form MOEA from 100 node data using an initial population which includes a few individuals from heuristic algorithm 2.

experiment; however, in general, quality of overall solutions is poorer for 100 node data.

In this case, we could appreciate the poorer quality of solutions in this region because we know the extreme solutions obtained by Algorithm 2. Had we not developed the heuristic and the solutions, we could not have any clue about the poorer quality of the obtained solution front in one part region of the solution space.

4.3 Improving upon the EMO results

We analyze the above experiment, and note that the poorer solution quality is attributed to the larger data set (100 node data) and in the smaller data set (50 node data). Also, the initial population was clustered much away from the lower-cost region, the final solutions have all been shifted to the lower-cost region. This indicates that, perhaps, the genetic operators were biased towards evolving minimal spanning tress. The other possible region to this phenomenon can be attributed to genetic operators becoming weak in shifting the individuals towards smaller-intersections region. There may be some other possible reasons too.

In an attempt to find solutions in the entire region, we injected a couple of individuals belonging to the higher-cost, lower-intersection region in the randomly generated population. Surprisingly, results were very encouraging; this results in spread of solutions. A plot showing the results for 100 node data set is included in Figure 6. It can be observed that the solutions obtained in Figure 6 are much superior to Figure 5, and cover the entire range of the feasible region.

The other possible strategy may be to re-design the genetic operators. However, since the quality is much improved simply by injecting a few selected individuals, we do not further investigate how to improve the solution quality as obtained by total randomly generated population. This is an area of future research.

Finally, we quantitatively evaluate the solution fronts obtained from each of the algorithms. We compute convergence [7], spread [6], hypervolume (S measure) [29], and coverage [29] measures. The *convergence* metric measures the convergence of the obtained solution set against a reference set. In the case of unknown problems, approximation sets of all the considered algorithms are combined and non-dominated approximation set is computed to act as a reference set. A lower value of the matrix indicates better convergence. Ideally, this should be zero. The *spread* metric also uses a reference set to measure the average distribution of points in obtained solution set and the distance between the extreme solutions in obtained solution set in comparison to extreme solutions in reference set. Here also, a lower value of the metric indicates better spread. The *hypervolume* metric provides the volume of search space covered by the obtained set of solutions with reference to a reference point. Hence, a higher value indicates superiority of solution (better solution). The *coverage* metric does not require a reference set rather it compares two set of solutions directly and provides the information that the solution set of one algorithm dominates the solution set of another algorithm, although it does not tell how much superior is one to the other. For a typical case, computed values are shown in Table 1; MOEA+ indicates MOEA strategy with a few special individuals injected during population initialization. The coverage metric gives good values too. These observations are similar to the visual inspection of the plots.

Table 1: Metrics computed for 100 node data.

Algorithm	Convergence	Spread	S Measure
Heuristics	0.0984	0.8791	23.0374
MOEA	0.0000	0.2414	54.2940
MOEA+	0.0000	0.0082	68.8504

5. CONCLUSIONS

In this work, we have attempted to solve a biobjective computationally hard problem having applications to VLSI chip design using evolutionary multiobjective optimizer. Since the problem is hard, and there do not exist any experimental results about the Pareto-front, we developed two heuristics, which give an indication about the extreme points in each objective space.

First, we obtained results from EMO without using much problem specific information. Results thus obtained are diverse and quite promising. However, on comparing with the extreme solutions obtained by heuristics, we observed that the EMO did not yield the complete Pareto-front. Therefore, we embed this information in problem solving strategy, and the results obtained are seemingly close to the true Pareto-front. This is a paradox in itself that for solving a problem *per se* we need to know the solution space before hand. In some recent work done independently, Ishibuchi and Narukawa [14] and Kumar et al. [21] observed that good solutions are not always obtained by MOEAs.

Therefore, apart from solving a hard problem, the other spin-off research from this work is: can we effectively solve *unknown* problems using black-box optimization technique of EA, especially in a multi-objective setting? How can one trust on the solutions obtained for real-world applications (RWAs) by such black-box optimization? Can we effectively approximate the *quality* of solutions?

6. REFERENCES

[1] M. Abellanas, J. Garcia, G. Hernandez, N. Noy, and P. Ramos. Bipartite embeddings of trees in the plane. In *Graph Drawing 96, LNCS 1990*, pages 1–10, 1997.

[2] J. Akiyama and J. Urrutia. Simple alternating path problems. *Discrete Mathematics*, 84:101–103, 1990.

[3] V. Bilo, V. Goyal, R. Ravi, and M. Singh. On the crossing spanning tree problem. In *Approx - Random*, pages 51–60, 2004.

[4] C.A.C. Coello and G.B. Lamont (Eds.). *Applications of Multiojective Evolutionary Algorithms.* World Scientific, 2004.

[5] T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein. *Introduction to Algorithms, 2e.* Boston, MA: MIT Press, 2001.

[6] K. Deb. *Multiobjective Optimization Using Evolutionary Algorithms.* Chichester, UK: Wiley, 2001.

[7] K. Deb and S. Jain. Running performance metrics for evolutionary multiobjective optimization. In *SEAL*, pages 13–20, Singapore, Nov. 2002.

[8] N. Deo and N. Kumar. Constrained Spanning Tree Problems: Approximate Methods and Parallel Computations. *Discrete Mathematics and Theoretical Computer Science*, 40:191 – 217, 1998.

[9] A. Dumitrescu and J. Pach. Partitioning colored point sets into monochromatic parts. *Int. J. Comput. Geom. Appl.*, 12(5):401–412, 2002.

[10] C.M. Fonseca and P.J. Fleming. Multiobjective optimization and multiple constraint handling with evolutionary algorithms - Part I: a unified formulation. *IEEE Trans. Systems, Man and Cybernetics-Part A: Systems and Humans*, 28(1):26 – 37, 1998.

[11] S. Har-Peled and P. Indyk. When crossing counts - approximating the minimum spanning tree. In *16th Annual Symposium on Computational Geometry*, pages 166–175, 2000.

[12] J. Ho, G. Vijayan, and C. Wong. A new approach to the rectilinear Steiner tree problem. *IEEE Trans. Computer Aided Design*, 9(2):185–193, 1985.

[13] F.K. Hwang. On Steiner minimal trees with rectilinear distance. *SIAM Journal of Applied Mathematics*, 30:104–114, January 1976.

[14] H. Ishibuchi and K. Narukawa. Comparison of evolutionary multiobjective optimization with reference solution-based single objective approach. In *GECCO*, pages 787 – 794, 2005.

[15] B.A. Julstrom and G.R. Raidl. Edge sets: An effective evolutionary coding of spanning trees. *IEEE Trans. Evolutionary Computation*, 7(3):225 – 239, 2003.

[16] A. Kaneko and M. Kano. Discrete geometry on red and blue points in the plane - a survey. *Discrete and Computational Geometry: Algorithms and Combinatorics*, 25:551–570, 2003.

[17] M. Kano, C. Merino, and J. Urrutia. On plane spanning trees and cycles of multicolored point sets with few intersections. *Information Processing Letters*, 93:301–306, 2005.

[18] J.D. Knowles and D.W. Corne. A new evolutionary approach to the degree-constrained minimum spanning tree problem. *IEEE Trans. Evolutionary Computation*, 4(2):125 – 133, 2000.

[19] J.D. Knowles and D.W. Corne. On metrics for comparing nondominated sets. In *CEC*, vol. I, pages 711–716, 2002, IEEE.

[20] R. Kumar and P.I. Rockett. Improved sampling of the Pareto-front in multiobjective genetic optimization by steady-state evolution: a Pareto converging genetic algorithm. *Evolutionary Computation*, 10(3):283 – 314, 2002.

[21] R. Kumar, P.K. Singh, A.P. Singhal, and A.Bhartia. Evolutionary and heuristic algorithms for 0-1 knapsack problem. In *10th Online World Conf. Soft Computing & Industrial Applications, Applications of Soft Computing: Recent Trends.* Springer, 2006.

[22] S. Majumder, B.B. Bhattacharya, and S.M.A. Jafri. Reducing crossing number of multi-color rectilinear Steiner trees using monochromatic partitioning. In *Technical Report*, ACMU/VLSI, Indian Statistical Institute, Kolkata, 2005.

[23] M.V. Marathe, R. Ravi, R. Sundaram, S.S. Ravi, D.J. Rosenkrantz, and H.B. Hunt. Bicriteria network design problems. *J. Algorithms*, 28(1):142 – 171, 1998.

[24] P. Mazumdar and E. Rudnick. *Genetic Algorithms for VLSI Design, Layout & Test Automation.* Prentice Hall PTR, 1999.

[25] J. Pach. Geometric graph theory. In *Surveys in Combinatorics, 1999 (Canterbury)*, volume 267, pages 167–200. London Math. Soc. Lecture Notes Ser., Cambridge University Press, Cambridge, 1999.

[26] G.R. Raidl and B.A. Julstrom. Greedy heuristics and an evolutionary algorithm for the bounded-diameter minimum spanning tree problem. In *ACM Symposium on Applied Computing (SAC)*, pages 747 – 752, 2003.

[27] N.A. Sherwani. *Algorithms for VLSI Physical Design Automation, 3e.* Kluwer, 1999.

[28] S. Tokunaga. Intersection number of two connected geometric graphs. *Information Processing Letters*, 59:331–333, 1996.

[29] E. Zitzler. Evolutionary Algorithms for Multiobjective Optimization: Methods and Applications. *PhD Thesis*, Swiss Federal Institute of Technology (ETH), Zurich, November 1999.

[30] E. Zitzler, L. Thiele, M. Laumanns, C.M. Fonseca, and V.G. da Fonseca. Performance assessment of multiobjective optimizers: An analysis and review. *IEEE Trans. Evolutionary Computation*, 7:117 – 132, 2003.