International Journal of Electrical and Computer Engineering (IJECE)
Vol. 10, No. 3, June 2020, pp. 2959~2968
ISSN: 2088-8708, DOI: 10.11591/ijece.v10i3.pp2959-2968

Memory and I/O optimized rectilinear Steiner minimum tree routing for VLSI

Latha N. R., G. R. Prasad
Department of CSE, B.M.S. College of Engineering, India

Article Info

Article history:

Received Jun 11, 2019
Revised Dec 16, 2019
Accepted Jan 7, 2020

Keywords:

Minimum spanning tree
Rectilinear Steiner tree
VLSI
Wirelength estimation

ABSTRACT

As the size of devices are scaling down at rapid pace, the interconnect delay play a major part in performance of IC chips. Therefore minimizing delay and wire length is the most desired objective. FLUTE (Fast Look-Up table) presented a fast and accurate RSMT (Rectilinear Steiner Minimum Tree) construction for both smaller and higher degree net. FLUTE presented an optimization technique that reduces time complexity for RSMT construction for both smaller and larger degree nets. However for larger degree net this technique induces memory overhead, as it does not consider the memory requirement in constructing RSMT. Since availability of memory is very less and is expensive, it is desired to utilize memory more efficiently which in turn results in reducing I/O time (i.e. reduce the number of I/O disk access). The proposed work presents a Memory Optimized RSMT (MORSMT) construction in order to address the memory overhead for larger degree net. The depth-first search and divide and conquer approach is adopted to build a Memory optimized tree. Experiments are conducted to evaluate the performance of proposed approach over existing model for varied benchmarks in terms of computation time, memory overhead and wire length. The experimental results show that the proposed model is scalable and efficient.

Corresponding Author:

Latha N. R.,
Department of CSE, B.M.S.,
College of Engineering, India.
Email: lathanr11@rediffmail.com

1. INTRODUCTION

Rectilinear Steiner Minimal Tree (RSMT) is composed of small set of connected pins through Steiner nodes with minimal cumulative edge size in Manhattan distance for a given set of pins. The construction of RSMT is a major issue in designing Very Large Scale Integration (VLSI) such as interconnects design, placement and floor planning. It has been adopted in computing transmission delay, interconnect delay and in workload computation. It is also adopted in some global routing strategies to build a routing topography of all nets.

The construction of RSMT for VLSI is considered a Non-deterministic polynomial problem [1], as a result rectilinear minimum spanning tree (RMST) has been adopted in some earlier design by exploring space dimensional design. RMST construction requires fast tree computing strategy and since the RMST does not allow Steiner nodes in tree construction the resulting RMST, length is longer than that of RSMT. In [2] showed that RMST is one and half times greater than that of RSMT with less than 50% in terms of accuracy, which is tolerable in earlier design. However, the later design requires good wire length accuracy for which the construction of RSMT is required. In [3] presented a wide range characteristic of RSMT construction. In [4, 5] presented an optimal strategy for RSMT construction, which is said to have least computation time. In [6] presented a near optimal solution for RSMT construction. However, they are computationally very heavy and are not suitable for applications, specifically for VLSI design.

Many approaches have been presented to reduce time complexity in constructing RSMT. In [7] adopted spanning graph [8] to aid in building the primary set of spanning tree and obtain finest sets for the edge-which are computed iteratively to eliminate longest edge. In [9] presented a greedy batched technique, which improved efficiency and reduced the computation time. The Single Trunk Steiner Tree (STST) is built to connect a set of pins to individual trunks, which traverse vertically or horizontally through set of all pins, but is not efficient for medium size pins. In [10] presented refined single-trunk tree for degree up to 5 nets and it is optimally accurate for medium degree nets with fair run time complexity. In [31, 34] spanning tree based approximation algorithm that produced optimal solutions were presented.

In [11, 12] presented lookup table based fast and accurate optimal solution for RSMT construction namely FLUTE. In this technique, the nets are recursively broken into sub set of nets. FLUTE is evaluated for low degree nets and it is suitable for VLSI design. FLUTE is also efficient for high degree nets with runtime complexity of $O(n \log n)$. However, for higher degree nets the accuracy of RSMT construction is severely affected. This is due to the error induced during net breaking technique. To address this issue in [13] presented a scalable net partitioning technique, where the nets are broken into smaller subset of nets and again merged by adding Steiner nodes. This technique could handle both smaller and larger degree nets with slight reduction in accuracy but it induced a runtime complexity of $O(n \log^2 n)$. In [14] presented a fast lookup table based RSMT construction, which brings a good tradeoff between accuracy and the runtime complexity. As specified in [28-30, 32] memory is gaining prominence and efficient use of this resource is very important. Both [13, 14] did not consider the memory constraint in building a look up table. The future VLSI design consists of fixed blocks such as IP blocks, macros, and so on and FLUTE is adopted by these researchers [15, 16]. In such designs minimizing wire length and reducing memory overhead is most desired. To address these issues the proposed work presents a memory optimized RSMT construction that reduces wire length and computational overhead complexities. The contribution of research work:

- No prior work has considered memory constraint in designing RSMT construction. The proposed work presents a memory optimized RSMT construction.
- The proposed model reduces the wire length and computation time in constructing RSMT.
- The proposed model is evaluated considering different benchmark [25] and shows that the proposed model is efficient considering all benchmark in terms of memory overhead, computation time and wire length.

The paper organization is as follows: In section 2 extensive literature survey is carried out. The proposed memory optimized RSMT models are presented in Section 3. The experimental study considering various benchmark are presented in penultimate section. The concluding remark and future work is discussed in the last section.

2. LITTERATURE SURVEY

VLSI is a technique of combining lakhs of transistors into solitary Integrated Circuit (IC) chip. With the increase in transistors, the interconnecting wire length also increases. It is challenging to minimize the resistive and capacitive features, which have an impact on delay. The interconnect wires have fixed width and area making length as the only parameter that can be optimized. As a result, many routing techniques have been proposed in VLSI designs that are as surveyed below.

In [17] showed that the global router generally decompose net through RSMT. Therefore, to reduce congestion and provide flexibility it mainly depends on RSMT construction. FLUTE is a widely adopted technique for fast RSMT construction with minimal wire length. However, it fails to incorporate congestion. To provide flexibility and congestion optimization for net [17] presented a model namely Fthu, which is a two-phase approach by adopting FLUTE. In first phase, it decreases congestion and increases flexibility by applying reformed edge shifting and edge shrinking technique without changing Steiner tree topology. In second phase, the congested Steiner tree is broken and reconnected using MST-based approach. The outcomes show better performance in terms of reduced congestion time. However, there is no improvement in wire length performance.

In [18, 19, 26, 27] presented a model to solve global routing problem. In [18] presented model, namely GRIP (Global Routing Technology via Linear Programming).This model presented integer-programming model for current large-scale network. The model obtained high quality solution by adopting FLUTE for initial RSMT construction. The outcome shows improvement in cost and wire length performance. However, they did not exploit CPU and memory performance. Linear programming model are prone to get stuck in local optima. To overcome [19] presented a fast congestion driven Steiner tree creation by adopting FLUTE. The outcome shows significant in terms of runtime complexity. To solve the congestion in global routing [20, 21, 33] adopted game theory approach. The game theory approach is adopted to improve runtime complexity of clustering approach for VLSI routing placement design.

In [22] studied various clustering based placement tool. An efficient clustering approach can aid in reducing wire length, cycle time or optimize a design based on these objectives. However clustering approach can induce time constraint. To address the time constraint [22] exploited a heterogeneous computing and presented a parallel clustering approach for placement. Their model is exploited for both CPU as well as GPU. The model utilizes the CPU and GPU core to full extent. The outcome shows it achieves a good speed up when compared to serial execution strategy. However adopting GPU for processing induces high cost of deployment and their model did not consider the memory constraint. As a result, it increases I/O access time.

Extensive literature survey carried out shows that minimizing time complexity (runtime) and wire length is a critical factor for designing an efficient routing technique in VLSI design. Some existing approaches have considered minimizing wire length or runtime and some considered both for optimization. To improve runtime few approaches have considered a parallel implementation by utilizing CPU and GPU core. However, none of the approaches has considered memory performance. Utilizing the memory efficiently can aid in reducing the time complexity (i.e. I/O access time). The proposed work presents a memory optimization based RSMT to improve wire length, runtime and memory performance. In the next section below the proposed memory optimized RSMT (MORSMT) model is presented.

3. PROPOSED MEMORY OPTIMIZED RSMT MODEL

Here we present a memory optimized RSMT construction that reduces wire length, memory usage and computation time. As similar to [14], let us consider that the size of each sub tree be divided based on memory optimized tree and takes memory and spanning tree as input. Firstly it computes the least overhead edges (using memory optimized spanning graph) and selects one of the node as its root. The node, which is closer to the root node, is considered as parent node by realizing child-parent relationship along each of the edges. Then depth-first search and divide and conquer approach is adopted to optimize memory for larger size nets. Let us consider a graph $H(N, M)$, where M and N depicts a set of ordered pairs of edges and nodes respectively. Let $m = |E|$ and $n = |V|$ represent set of edges and nodes respectively. Here, we first construct an initial spanning graph H by adding Steiner nodes α and is considered to be connected to all nodes in H. Then divide-conquer approach is applied to build a memory optimized tree of graph H. Below table shows the notations and symbols used in the paper.

Table 1. Notations and symbols used

Symbol used	Abbreviation
H	Graph
N	Set of nodes
M	Set of edges
$H(N\|M)$	It is a directed graph
G	Spanning graph
α	Set of Steiner nodes
S	Memory Available
μ	Memory optimized subtree graph
d	Number of subtree

3.1. Memory optimized divide and conquer approach

The memory optimized divide conquer approach takes memory S, Spanning graph G of H and graph H as an input and obtain a tree Gas output which is a depth first search tree of H, where G is retained in memory and H is kept in disk. The algorithm first tests whether graph H can satisfy memory optimization requirement, so that the H can be loaded into S and if so it computes memory optimized tree G of H using available-memory optimization strategy and obtains G. Or else if does not obtain any G, the algorithm further computes memory optimized tree G of H by dividing memory optimized tree G by using divide and conquer approach.

To obtain an efficient memory optimized tree the legal dividend of H must be computed which is set to false initially as shown in flowchart in Figure 1. Then the present spanning graph G is optimized with respect to H until G is a memory optimized tree G of H or we obtain a legal dividend of H on spanning tree

G. Here the dividend is obtained by invoking dividend optimization technique to achieve a graph division H_0,H_1,H_2,…,H_d of H with resultant spanning graph⟦G⟧_0,G_1,G_2,…,G_d. The dividend optimizer also evaluates a memory optimized graph μ during the merge operation.

The dividend is said to be legal only if d>1 as shown in flow chart in Figure 1. Once the legal tree division is obtained, the memory-optimized tree G_q is computed for all sub-graph H_q using divide and conquer approach in a recursive manner. Then by combining all memory optimized tree G_q of H_q based on μ the memory optimized tree G is computed and obtain G as memory optimized tree of H. The overall flow of proposed memory optimized RSMT construction is shown in Figure 1.

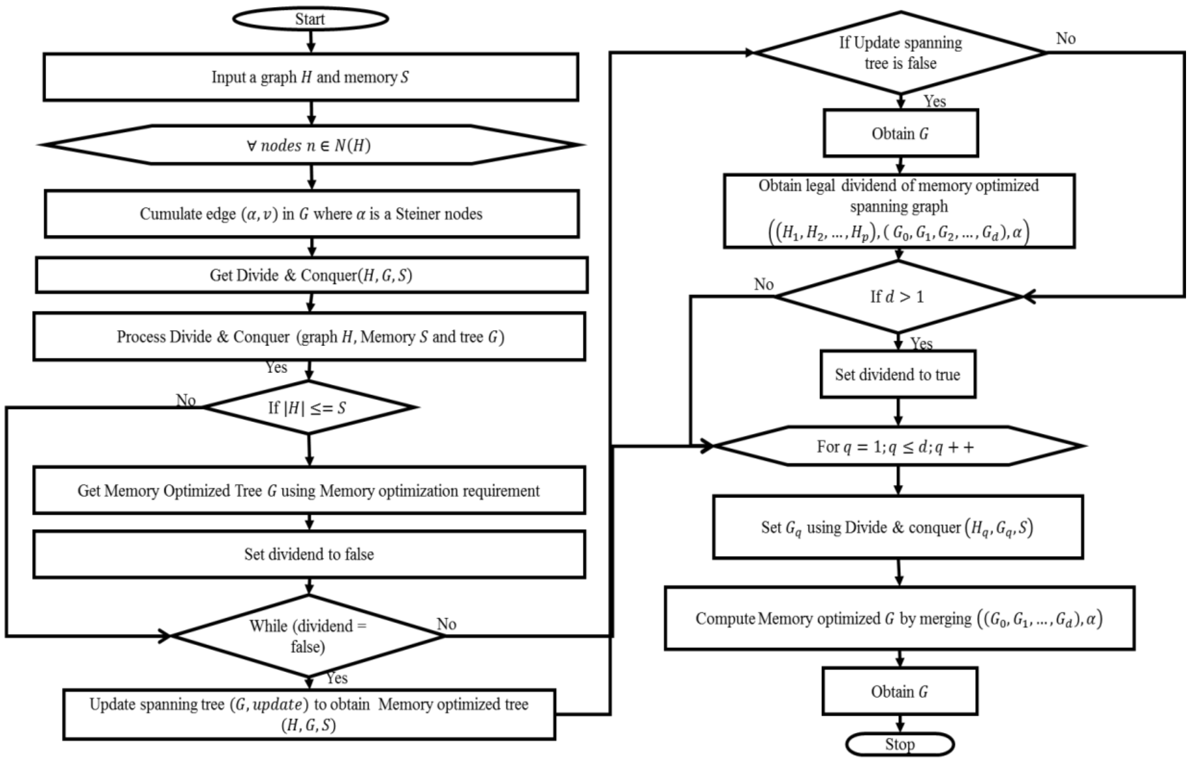
Start
Input a graph H and memory S
$\forall$ nodes $n \in N(H)$
Cumulate edge (α, v) in G where α is a Steiner nodes
Get Divide & Conquer(H, G, S)
Process Divide & Conquer (graph H, Memory S and tree G)
Yes
No
If $|H| \leq= S$
Get Memory Optimized Tree G using Memory optimization requirement
Set dividend to false
While (dividend = false)
No
Yes
Update spanning tree $(G, update)$ to obtain Memory optimized tree (H, G, S)
If Update spanning tree is false
No
Yes
Obtain G
Obtain legal dividend of memory optimized spanning graph $((H_1, H_2, \dots, H_p), (G_0, G_1, G_2, \dots, G_d), \alpha)$
No
If $d > 1$
Yes
Set dividend to true
For $q = 1; q \leq d; q++$
Set G_q using Divide & conquer (H_q, G_q, S)
Compute Memory optimized G by merging $((G_0, G_1, \dots, G_d), \alpha)$
Obtain G
Stop

Figure 1. Memory optimized based rectilinear steiner minimum tree construction

3.2. Memory optimized division algorithm

The objective of memory optimized divide and conquer approach is to maximize the number of divided subgraph. In existing model, for a given spanning tree G and graph H, the division is obtained using structure G_0 with same parent as G. This leads to following problem. Firstly, μ is obtained on top of G_0, where G_0 is generated based on only one level of nodes in G. The relationship of subgraph induced by subtrees rooted at leaf nodes of a_0 is complex or when the parent a_0 at G has limited number of child nodes, after evaluating the division by contracting all SCCs (Strongly Connected Component), this might result in availability of only few divided subgraphs. Secondly, μ is obtained by scanning graph H on disk once and evaluate set of edges$\bar{\mu}$, namely $\bar{\mu}(\bar{a}_q, \bar{a}_r)$ with $\bar{a}_q$ and $\bar{a}_r$ be the leaf node of a_0 in G, whereas, the number of leaf node a_0 is less, then $\bar{\mu}$ may be smaller than the$\bar{\mu}$, which is available in graph. As a result large amount of $\bar{\bar{\mu}}$ is computed but not utilized during scanning edges. This reduces the I/O efficiency, which results in the increase in computation time.

Our proposed model will overcome these problems by enlarging the size of G_0 and its correspondent μ with respect to memory size (i.e. whether they can fit in main memory). To satisfy memory constraint, the model considers multiple levels of nodes in G to generate G_0 and it's correspondent μ. The multi-level subtree G is defined as a partitioned treeG_p. Let us consider a tree G with parent nodea_0, partitioned tree $G_p$that is a subtree of Gmust satisfy the following condition. Firstly, the parent of G_p should be G_0. Secondly, for any node y, for instance the leaf nodes of y in G are $y_1, y_2, y_3, \dots, y_n$, if $y \in N(G_p)$, then y is either a node in G_p with leaf nodes $y_1, y_2, y_3, \dots, y_n$ or a leaf node of y inG_p.

To satisfy these constraints, consider a tree G with parent a_0 and memory constraint $\overline{S}$, the partitioned tree G_p is generated as follows. The G_p initially is composed of one node a_0. Then the child node y are iteratively selected from G_p, where all leaf nodes of y in G as the leaf nodes of y in G_p. Note μ with respect to G_p comprises of at least $\left|N\left(G_p\right)\right|^2$ edges. As a result, the execution is stopped when adding node $\left|N\left(G_p\right)\right|^2 > S$.

The memory optimized division model is as presented in Figure 2. Here G_0 is constructed in top-down fashion based on G_p. The algorithm first evaluates partition tree $\bar{G}_0$ of G using above discussed method and initialize μ to be $\bar{G}_0$. Then it searches all edges (x, y) in H on disk and add $\left(u_x, u_y\right) = \mu(x, y)$ into μ, if u_x and u_y belongs to $N(\bar{G}_0)$. Then the model finds all $\mathcal{T}$ in μ and top-down methodology is used to generate G_0. After that G_0 and FIFO (First in First out) queue Q is initialized. It first pushes parent a_0 of G into Q. Then the edges are iteratively added into G_0 until Q becomes null. In every round, it first retrieves the top node x in Q and pushes all leaf nodes y of x into G_0 (i.e. if y is in the tree and x is not a Steiner node). For each such instances (i.e. y pushed into G_0), it is further pushed into Q for further expansion. Once G_0 is computed, μ is updated. The μ is updated by popping (deleting) all nodes that are not in $N(G_0)$ from μ. Lastly, divided subgraph $H_1, H_2, H_3, \dots, H_d$ and subtrees $G_1, G_2, G_3, \dots, G_d$ are evaluated.

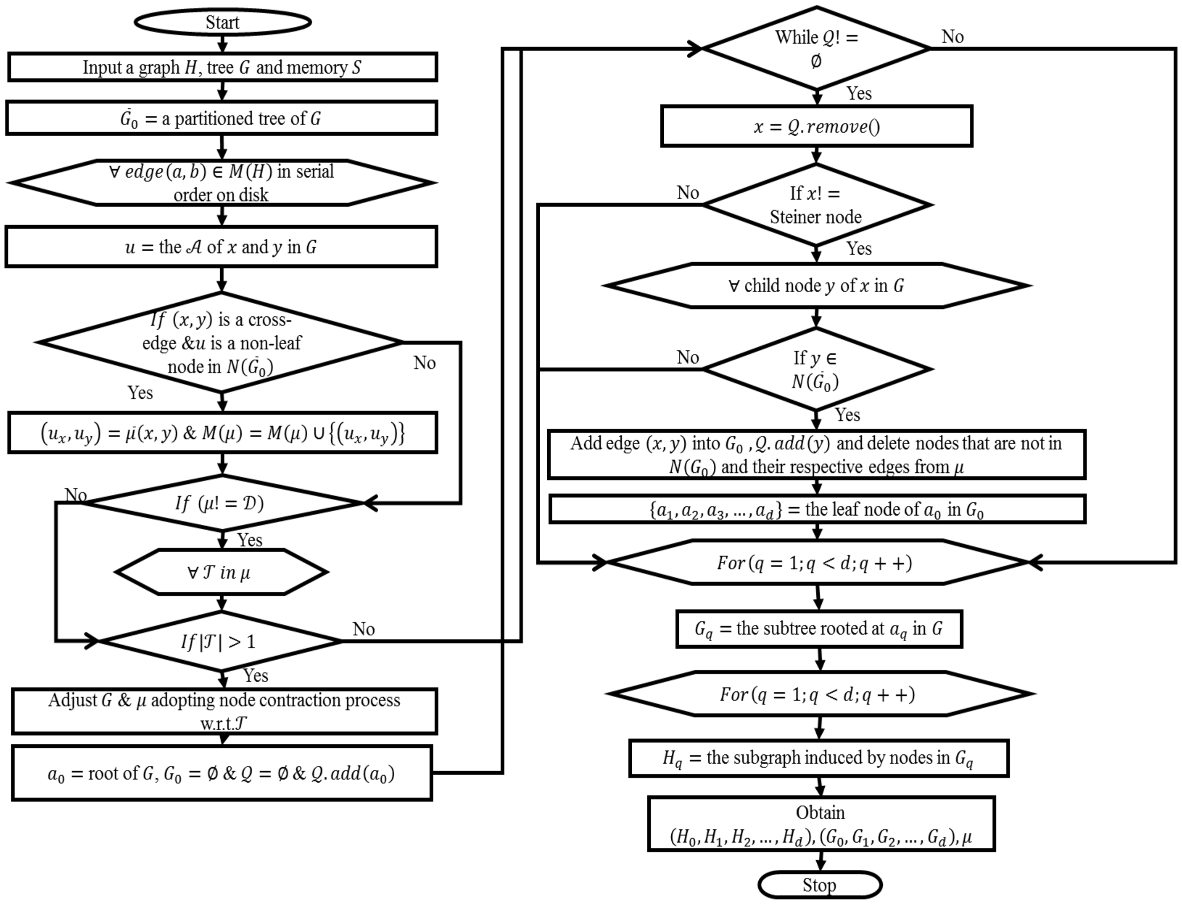
Start
Input a graph H, tree G and memory S
$\bar{G}_0$ = a partitioned tree of G
$\forall\ edge(a, b) \in M(H)$ in serial order on disk
u = the $\mathcal{A}$ of x and y in G
$If\ (x, y)$ is a cross-edge &u is a non-leaf node in $N(\bar{G}_0)$
No
Yes
$(u_x, u_y) = \mu(x, y)$ & $M(\mu) = M(\mu) \cup \{(u_x, u_y)\}$
No
$If\ (\mu! = \mathcal{D})$
Yes
$\forall\ \mathcal{T}\ in\ \mu$
$If |\mathcal{T}| > 1$
No
Yes
Adjust G & μ adopting node contraction process w.r.t.$\mathcal{T}$
a_0 = root of G, $G_0 = \emptyset$ & $Q = \emptyset$ & $Q.add(a_0)$
While $Q! = \emptyset$
No
Yes
$x = Q.remove()$
No
If $x! =$ Steiner node
Yes
$\forall$ child node y of x in G
No
If $y \in N(\bar{G}_0)$
Yes
Add edge (x, y) into G_0 ,$Q.add(y)$ and delete nodes that are not in $N(G_0)$ and their respective edges from μ
$\{a_1, a_2, a_3, \dots, a_d\}$ = the leaf node of a_0 in G_0
$For(q = 1; q < d; q++)$
G_q = the subtree rooted at a_q in G
$For(q = 1; q < d; q++)$
H_q = the subgraph induced by nodes in G_q
Obtain $(H_0, H_1, H_2, \dots, H_d), (G_0, G_1, G_2, \dots, G_d), \mu$
Stop

Figure 2. Memory optimized division algorithm

3.2. Memory optimized merging algorithm

The merge algorithm presented in Figure 3 takes as input, a divided tree $G_0, G_{1,} G_2, \dots, G_d$ and the corresponding μ and outputs a graph G. To perform the merge operation according to the algorithm in Figure 3, the following issues must be solved. First issue is how to arrange $G_0, G_{1,} G_2, \dots, G_d$ in the merged tree G, such that G is a tree of graph H. And Second issue is how to handle the Steiner node in tree $G_0, G_{1,} G_2, \dots, G_d$. The flow of merging algorithm is as shown in Figure 3.

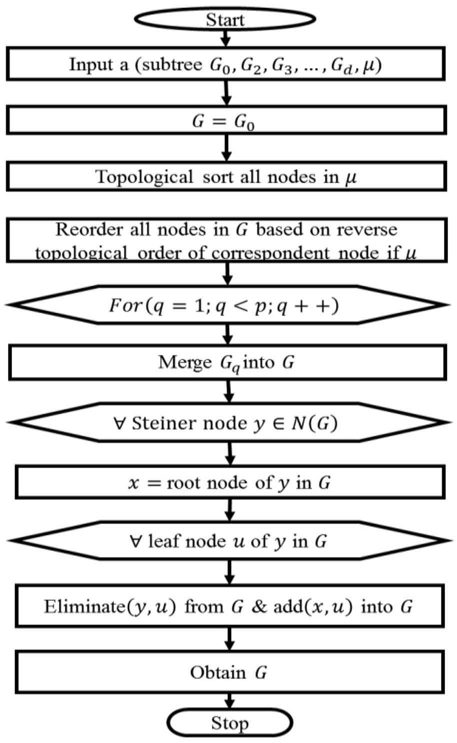
Start
Input a (subtree $G_0, G_2, G_3, \dots, G_d, \mu$)
$G = G_0$
Topological sort all nodes in μ
Reorder all nodes in G based on reverse topological order of correspondent node if μ
$For(q = 1; q < p; q++)$
Merge G_q into G
∀ Steiner node $y \in N(G)$
x = root node of y in G
∀ leaf node u of y in G
Eliminate(y, u) from G & add(x, u) into G
Obtain G
Stop

Figure 3. Memory optimized merging algorithm

To solve the first issue, we use information of μ (i.e. μ is a graph that preserves the topology of edges of all partitioned subgraphs). Then sort all the nodes in μ and rearrange the nodes in G_0 based on reverse topological order of correspondent nodes in μ and then merge all $G_q(1 \leq q \leq d)$ with $G_0$to obtain tree of H, we need to be assured that μ is a DAG and $N(\mu) = N(G_0)$. To solve the second issue, we merge all trees$G_0, G_{1,} G_2, \dots, G_d$, to obtain a tree G. For each Steiner node $y \in N(G_0)$, for instance root node of y in G is x, then delete edge (x, y) from G, and for each leaf node u of y in G, we eliminate edge (y, u) from G and add edge (x, u) into G. This method aids in improving to validate that the resultant tree G is tree of H. The performance study of the proposed approach is presented in the next section.

4. RESULT AND ANALYSIS

The MORSMT algorithm is implemented using C++ object oriented programming language. The GCC compiler is used to compile the code. The eclipse Kepler IDE used for running the algorithm. The system environment used to run the algorithm is Centos 7.0 Linux operating system, 3.2 GHz, Intel I-5 Quad core processor and 16GB RAM. The IBM benchmark [25] is considered for evaluation, which is as shown in Table 2. The experiment is carried out to evaluate the performance of MORSMT over existing approach [14] in terms of wirelength, memory utilization and computation time.

4.1. Wirelength performance

Experiments are carried out to evaluate wirelength performance and 18 IBM circuit in ISPD98 benchmark is used. The information of benchmark is shown in Table 2. and there are 1.57 million nets in total. The proposed MORSMT approach is compared with FLUTE [14] with default accuracy $A = 3$ in terms of wirelength performance, which is shown in Table 3. The outcome shows that MORSMT performs better than existing approach in terms of wirelength reduction for all the cases. An average reduction of 0.026% is achieved by MORSMT over existing approach.

4.2. Computation time performance

The proposed MORSMT approach is compared with FLUTE [14] with default accuracy $A = 3$ in terms of computation time performance, which is shown in Table 4. The outcome shows that MORSMT performs better than existing approach in terms of computation time reduction for all the cases. An average improvement of 32.62% is achieved by MORSMT over existing approach.

Table 2. Benchmark details

Benchmark circuit case	Number of nets	Maximum degree	Average degree
IBM1	14111	42	3.58
IBM2	19584	134	4.15
IBM3	27401	55	3.41
IBM4	31970	46	3.31
IBM5	28446	17	4.44
IBM6	34826	35	3.68
IBM7	48117	25	3.65
IBM8	50513	75	4.06
IBM9	60902	39	3.65
IBM10	75196	41	3.96
IBM11	81454	24	3.45
IBM12	77240	28	4.11
IBM13	99666	24	3.58
IBM14	152772	33	3.58
IBM15	186608	36	3.84
IBM16	190048	40	4.10
IBM17	189581	36	4.54
IBM18	201920	66	4.06
Average	106299	134	3.92

Table 3. Wirelength performance

Benchmark circuit case	MORSMT	FLUTE [14]
IBM1	444307	444553
IBM2	527382	527641
IBM3	761993	762276
IBM4	855986	856273
IBM5	2809615	2810816
IBM6	494144	495969
IBM7	994978	995265
IBM8	944096	944382
IBM9	1260914	1261199
IBM10	3190871	3191615
IBM11	1898961	1899367
IBM12	2914884	2915521
IBM13	2450087	2450577
IBM14	3180260	3180777
IBM15	2922395	2922778
IBM16	3500272	3500776
IBM17	5368916	5369659
IBM18	2145856	2146128
Average	2036995.389	2037531.778

Table 4. Computation time performance

Benchmark circuit case	MORSMT	FLUTE [14]
IBM1	90000	180000
IBM2	130000	220000
IBM3	163700	267000
IBM4	161100	269000
IBM5	410000	870000
IBM6	183000	277000
IBM7	193000	298800
IBM8	250000	320000
IBM9	219000	322000
IBM10	260000	347000
IBM11	298000	390000
IBM12	390000	510000
IBM13	410000	570000
IBM14	520000	640000
IBM15	517000	651000
IBM16	587000	690000
IBM17	1190000	1780000
IBM18	1090000	1880000
Average	392322.2	582322.2222

4.3. Memory utilization performance

To evaluate the performance of memory usage, valgrind [23, 24] has been used. The proposed MORSMT approach is compared with FLUTE [14] with default accuracy $A = 3$ interms of memory utilization performance which is as shown in Table 5. The outcome shows that MORSMT performs better than existing approach in terms of memory consumption for all the cases. An average reduction of 77.71% is achieved by MORSMT over existing approach. The outcome shows that memory usage is directly dependent on wire length and degree size.

Table 5. Memory utilization performance

Benchmark circuit case	MORSMT	FLUTE [14]
IBM1	109,264	398,908
IBM2	168,457	713,451
IBM3	180,996	711,893
IBM4	211,003	723,607
IBM5	224,661	1,188,898
IBM6	244,577	970,098
IBM7	337,674	1,248,799
IBM8	414,185	2,078,590
IBM9	411,154	1,595,039
IBM10	524,740	2,239,228
IBM11	531,886	1,789,611
IBM12	543,178	2,407,032
IBM13	659,237	2,551,928
IBM14	1,030,486	3,801,149
IBM15	1,284,495	5,603,960
IBM16	1,348,313	5,732,675
IBM17	1,414,806	6804778
IBM18	1,627,262	6982462
Average	625,910	2,641,228

5. CONCLUSION

This work presented a memory efficient RSMT construction. The proposed model is an improvement of original FLUTE. FLUTE does not consider memory optimization in RSMT construction and adopted breadth first search to find minimum spanning tree, which induced memory overhead. To address this problem the proposed work adopts divide and conquer and depth first search to find the minimum spanning tree. The experiments are conducted to evaluate the performance of proposed approach over existing approach for varied benchmarks. The outcome shows significant performance improvement of 0.026%, 76.3%, and 32.62% over existing approach in terms of wirelength, memory overhead, and computation time reduction respectively. The future work would consider presenting parallel memory optimized RSMT construction and experiment will be carried out on multi-core environment such as CPU or GPU to improve speedup performance.

ACKNOWLEDGEMENTS

The author would like to acknowledge and thank Technical Education Quality Improvement Program [TEQIP] Phase 3, BMS College of Engineering.

REFERENCES

[1] M. R. Garey and D. S. Johnson, “Computers and intractability: A guide to the theory of NP-completeness,” New York: Freeman, 1979.
[2] F. K. Hwang, “On Steiner minimal trees with rectilinear distance,” *SIAM J. Appl. Math*., vol. 30, no. 1, pp. 104–114, Jan 1976.
[3] F. K. Hwang, D. S. Richards, and P. Winter, “The Steiner tree problem,” *in Annals of Discrete Mathematics*, Amsterdam, The Netherlands: Elsevier, 1992.
[4] D. M. Warme, P. Winter, and M. Zachariasen, “Exact algorithms for plane Steiner tree problems: A computational study,” *in Advances in Steiner Trees*, D. Z. Du, J. M. Smith, and J. H. Rubinstein, Eds. Norwell, MA: Kluwer, pp. 81–116, 2000.
[5] “GeoSteiner—Software for Computing Steiner Trees,” [Online]. Available: http://www.diku.dk/geosteiner
[6] V. Vani and G. R. Prasad, “An improved augmented line segment based algorithm for the generation of rectilinear Steiner minimum tree,” *International Journal of Electrical and Computer Engineering (IJECE)*, vol. 7, no. 3, pp. 1262-1267, June 2017.

[7] H. Zhou, “Efficient Steiner tree construction based on spanning graphs,” *in Proc. Int. Symp. Phys.*, pp. 152–157, 2003.

[8] H. Zhou, N. Shenoy, and W. Nicholls, “Efficient spanning tree construction without Delaunay triangulation,” *Inf. Process. Lett.*, vol. 81, no. 5, pp. 271–276, 2002.

[9] A. Kahng, I. Mandoiu, and A. Zelikovsky, “Highly scalable algorithms for rectilinear and octilinear Steiner trees,” *in Proc. Asian South Pacific Des. Autom. Conf.*, pp. 827–833, 2003.

[10] H. Chen, C. Qiao, F. Zhou, and C.-K. Cheng, “Refined single trunk tree: A rectilinear Steiner tree generator for interconnect prediction,” *in Proc. ACM Int. Workshop Syst. Level Interconnect Prediction*, pp. 85–89, 2002.

[11] C. Chu, “FLUTE: Fast lookup table based wirelength estimation technique,” *In Proc. IEEE/ACM Intl. Conf. on Computer-Aided Design*, pp. 696–701, 2004.

[12] C. Chu and Y.-C. Wong, “Fast and accurate rectilinear Steiner minimal tree algorithm for VLSI design,” in *Proc. Int. Symp. Phys*, pp. 28–35, Des 2005.

[13] Y-C. Wong and C. Chu, “A scalable and accurate rectilinear Steiner minimal tree algorithm*,” IEEE International Symposium on VLSI Design, Automation and Test (VLSI-DAT)*, pp. 29-34, 2008.

[14] C. Chu and Yiu-Chung Wong, “FLUTE: Fast lookup table based rectilinear Steiner minimal tree algorithm for VLSI design,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 27, no. 1, pp. 70-83, 2008.

[15] H. Zhanga, D-Y. Yea, W-Z. Guo, “A heuristic for constructing a rectilinear Steiner tree by reusing routing resources over obstacles,” *Integration*, vol. 55, pp. 162–175, 2016.

[16] P. P. Saha, S. Saha and T. Samanta, “An efficient intersection avoiding rectilinear routing technique in VLSI,” *International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, Mysore, pp. 559-562, 2013.

[17] K. Ma, Q. Zhou, Y. Cai, C. Zhang, and Z. Qi, “A Steiner tree construction method for flexibility and congestion optimization,” *International Conference on Communications, Circuits and Systems (ICCCAS)*, Chengdu, pp. 519-523, 2013.

[18] T. H. Wu, A. Davoodi, and J. T. Linderoth, “GRIP: Global routing via integer programming,” in *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 30, no. 1, pp. 72-84, Jan 2011.

[19] W. W. Kai, N. Ahmad, M. H. Jabbar, “Variable body biasing (VBB) based VLSI design approach to reduce static power,” *International Journal of Electrical and Computer Engineering (IJECE)*, vol. 7, no. 6, pp. 3010-3019, Dec 2017.

[20] U. F. Siddiqi, S. M. Sait, and Y. Shiraishi, “A game theory-based heuristic for the two-dimensional VLSI global routing problem,” *Journal of Circuits Systems and Computers*, vol. 24, no. 6, pp. 1550082:1-1550082:19, 2015.

[21] U. F. Siddiqi and S. M. Sait, “A game theory based post-processing method to enhance VLSI global routers,” *IEEE Access*, vol. 5, pp. 1328–1339, 2017.

[22] K. V. Rajkumar, A. Yesubabu, and K. Subrahmanyam, “Fuzzy clustering and fuzzy C-means partition cluster analysis and validation studies on a subset of CiteScoredataset,” *International Journal of Electrical and Computer Engineering (IJECE)*, vol. 9, no. 4, pp. 2760-2770, Aug 2019

[23] J. Seward and N. Nethercote, “Using valgrind to detect undefined value errors with bit-precision,” in *Proc. of the USENIX Annual Technical Conference*, pp. 2–2, 2005.

[24] N. Nethercote, R. Walsh, and J. Fitzhardinge, “Building workload characterization tools with valgrind,” *IEEE International Symposium on Workload Characterization*, San Jose, CA, pp. 2-2, 2006.

[25] C. J. Alpert, “The ISPD98 circuit benchmark suite,” in *Proc. Int. Symp. Phys.*, pp.80–85, Des 1998. [Online]. Available: http://vlsicad.ucsd.edu/UCLAWeb/cheese/ispd98.html

[26] Y. Han, K. Chakraborty, and S. Roy, “A global router on GPU architecture,” in *IEEE International Conference on Computer Design (ICCD)*, pp. 1–6, 2013.

[27] H. Shojaei, A. Davoodi, and J. Linderoth, “Congestion analysis for global routing via integer programming,” in *IEEE International Conference on Computer-Aided Design (ICCAD)*, pp. 256–262, 2011.

[28] G. Bosilca, A. Bouteiller, A. Danalis, M. Faverge, T. Herault, and J. J. Dongarra, “PaRSEC: Exploiting heterogeneity for enhancing scalability,” *Computing in Science & Engineering*, vol. 15, no. 6, pp. 36–45, 2013.

[29] E. Agullo, P. R. Amestoy, A. Buttari, A. Guermouche, J. L’Excellent, and F. Rouet., “Robust memory-aware mappings for parallel multifrontal factorizations,” *SIAM J. Scientific Computing*, vol. 38, no. 3, 2016.

[30] G. Aupy, C. Brasseur, and L. Marchal, “Dynamic memory-aware task-tree scheduling,” In *Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS)*, IEEE, pp. 758–767, 2017.

[31] M. Jacquelin, L.Marchal, Y. Robert, and B. Uçar, “On optimal tree traversals for sparse matrix factorization,” In *Parallel & Distributed Processing Symposium (IPDPS), 2011 IEEE International*, IEEE, pp. 556–567, 2011.

[32] M. Sergent, D. Goudin, S. Thibault, and O. Aumage, “Controlling the memory subscription of distributed applications with a task-based runtime system,” In *Proceedings of the International Parallel and Distributed Processing SymposiumWorkshops*, pages 318–327. IEEE, 2016.

[33] K. Ma, Q. Zhou, Y. Cai, C. Zhang and Z. Qi, "A Steiner tree construction method for flexibility and congestion optimization," *2013 International Conference on Communications, Circuits and Systems (ICCCAS)*, Chengdu, pp. 519-523, 2013.

[34] Kasneci, G., Ramanath, M., Sozio, M., Suchanek, F.M., Weikum, G., “STAR: Steiner-tree approximation in relationship graphs,” In: *Proc. of IEEE International Conference on Data Engineering, IEEE*, Washington DC, USA, pp. 868-879, 2009.

BIOGRAPHIES OF AUTHORS

Latha N. R. is an Assistant Professor in the Department of Computer Science and Engineering, B.M.S. College of Engineering, Bangalore. She received B.E degree in Information Science and Engineering from Visvesvaraya Technological University in 2005 and M.Tech degree in Computer Science and Engineering from VTU in 2009. She is currently pursuing Ph.D. degree in VTU in the area of Parallel Computing.

Dr. Prasad G R is a Professor in Department of Computer Science & Engineering, BMSCE, Bangalore. He holds a Ph.D from National Institute of Technology, Karnataka, Surathkal, INDIA. He received his M.Tech degree in Computer Science & Engineering from Bangalore University in 1999 and B.E Degree in Computer Science & Engineering from Bangalore University in 1995. His research interests include Reconfigurable computing, Computing Accelerators.