Journal of Circuits, Systems, and Computers
Vol. 24, No. 6 (2015) 1550082 (19 pages)

DOI: 10.1142/S0218126615500826

World Scientific
www.worldscientific.com

A Game Theory-Based Heuristic for the Two-Dimensional VLSI Global Routing Problem*

Umair F. Siddiqi

Center for Communications & Information Technology Research,
Research Institute, King Fahd University of Petroleum & Minerals,
Dhahran 31261, Saudi Arabia
ufarooq@kfupm.edu.sa

Sadiq M. Sait†

Center for Communications & Information Technology Research,
Department of Computer Engineering,
College of Computer Science & Engineering,
King Fahd University of Petroleum & Minerals,
Dhahran 31261, Saudi Arabia
sadiq@kfupm.edu.sa

Yoichi Shiraishi

Department of Mechanical Science and Technology,
School of Science and Technology, Gunma University,
29-1 Honcho, Ota-Shi, Gunma 373-0052, Japan
yoichi.siraisi@gunma-u.ac.jp

Received 10 June 2014
Accepted 9 February 2015
Published 19 March 2015

In this work we propose a game theory (GT)-based global router. It works in two steps: (i) Initial routing of all nets using maze routing with framing (MRF) and (ii) GT-based rip-up and reroute (R&R) process. In initial routing, the nets are divided into several small subsets which are routed concurrently using multithreading (MT). The main task of the GT-based R&R process is to eliminate congestion. Nets are considered as players and each player employs two pure strategies: (attempt to improve its spanning tree, and, do not attempt to improve its spanning tree). The nets also have mixed strategies whose values act as probabilities for them to select any particular pure strategy. The nets which select their first strategy will go through the R&R operation. We also propose an algorithm which computes the mixed strategies of nets. The advantage of using GT to select nets is that it reduces the number of nets and the number of iterations in the R&R process. The performance of the proposed global router was evaluated on ISPD'98 benchmarks and compared with two recent global routers, namely, Box Router 2.0 (configured for speed) and Side-winder. The results show that the proposed global router with

*This paper was recommended by Regional Editor Emre Salman.
†Corresponding author.

MT has a shorter runtime to converge to a valid solution than that of Box Router 2.0. It also outperforms Side-winder in terms of routability. The experimental results demonstrated that GT is a valuable technique in reducing the runtime of global routers.

Keywords: Global routing; game theory; rip-up and rerouting; VLSI physical design.

1. Introduction

Global routing in a critical step in the physical design of integrated circuits and is an NP-hard problem.[1] It lies between the placement and detailed-routing steps in the physical design of VLSI chips. In global routing, the nets of wires are mapped to a coarse grid of global routing cells (or gcells). Each gcell has a fixed horizontal and vertical capacity. The task of global routing is to assign the nets while satisfying the capacity constraints of gcells.[2,3] Each net is routed by generating a spanning tree for it that covers all of its pins. A solution of the global routing problem which does not violate the capacity constraints of gcells is called a valid solution. Routability-driven (RD) placement is a recent development that uses global routing to guide the placement process.[4,5] The RD placement process will need to execute global routing several times. Therefore, the global router should not only be fast but must also yield good results.

Among the methods of routing, maze routing is the only method that guarantees to find a path between any two pins if there exists one. Therefore, many global routers use maze routing exclusively, or use other methods for initial routing and use maze routing for difficult-to-route nets.[6,7] However, maze routing is slow and memory intensive. Maze routing is generally used with a rip-up and reroute (R&R) process to produce valid solutions. The main task of the R&R process is to selectively R&R a small fraction of nets in order to eliminate congestion. Maze routing with framing (MRF) is a modification of maze routing in which a net determines its spanning tree within a bounding box on the grid.[2] The MRF method is very fast as compared to traditional maze routing and the size of the bounding box can be increased or decreased. Lee algorithm[2] is a popular method of maze routing and has a breath-first behavior. Recent research showed that Lee algorithm is highly suitable for implementation using parallel computing platforms such as graphics processor units (GPUs)[6,8–10] because of its simple data-structure and breath-first behavior which can be easily implemented on parallel platforms. A review of recently proposed global routers can be found in Ref. 11.

This work proposes a simple global router that uses MRF method exclusively. It also contains a R&R process whose role is to eliminate congestion. The R&R process is modeled using game theory (GT)[12,13] in which nets act as players that want to optimize their spanning trees in terms of congestion and wire-length. GT is a very useful technique of decision making and multi-agent optimization.[12,13] In the literature, GT has been successfully used in routing problem of communication networks[14] and multi-agent optimization problem.[15] GT is considered a viable tool in

wireless networking to solve routing issues.[16] However, it is the first time that GT is applied to the global routing problem.

The performance of the proposed global router was evaluated on industrial benchmarks ISPD'98.[17,a] The proposed global router is compared with two recent integer linear programming (ILP)-based global routers because they share a common property of employing optimization to eliminate congestion. The selected global routers are: (i) Box Router 2.0[18,19] whose parameters were configured for speed and (ii) Side-winder.[20,21] Box Router 2.0 has won third place in the ISPD 2007 global routing contest.[19] It employs a method of routing in which a smaller region (referred as box) is selected on the routing grid and nets that lie within the box are routed and congestion is minimized (or dispersed) using ILP. The box is initiated on the most congested region of the routing grid and gradually expands to cover the entire routing grid. It employs pattern routing and adaptive maze routing to route the nets. Side-winder is another recent global router in which the ILP technique is global and covers the whole grid.

The experimental results show that the proposed global router can solve all problems of ISPD'98 suite. It is fast as well as it can produce smaller wire-length. When compared with the speed optimized version of Box Router 2.0,[18,19] it has better runtime and wire-length. When compared with Side-winder,[20,21] it has better routability because Side-winder did not solved all problems of the ISPD'98 suite.

This paper is organized as follows. Section 2 describes the global routing problem. Section 3 describes the design of the proposed global router and its GT-based heuristic. Section 4 discusses the experimental results. Section 5 contains the conclusion.

2. Global Routing Problem

The global routing problem is modeled using a two-dimensional (2D) grid-graph G. The grid-graph has a set of vertices V and a set of edges E. Each vertex $v_i \in V$ corresponds to a gcell, and each edge $e_{ij} \in E$ corresponds to a boundary between adjacent vertices v_i and v_j. Each edge e_{ij} has a capacity c_{ij} which is the maximum number of nets or wires that can pass through it. The actual number of nets that are passing through an edge e_{ij} is called its demand and is represented as u_{ij}. The problem also contains a set of nets N, where each net $n_i \in N$ is composed of a set P_i of pins (with each pin corresponding to a vertex v_i). Each net $n_i \in N$ is routed by finding a tree t_i that covers all its pins. Each tree $t_i \in E$ is a set of edges without any cycles or repetition of edges and $t_i \subseteq E$. The set T stores the spanning trees of all nets.

The primary objective of global routing is to route all the nets and to make sure that the capacity constraints of the edges are not violated, i.e., $u_{ij} \leq c_{ij}, \forall e_{ij} \in E$. The edge e_{ij} is said to be congested or have an overflow if $u_{ij} > c_{ij}$. The edges whose

[a] http://cseweb.ucsd.edu/~kastner/labyrinth_vault/benchmarks/index.html (accessed on 30 December, 2014).

demand is equal to their capacity are said to be fully utilized. The fully utilized and congested edges are considered important in eliminating congestion. The congested edges contribute directly to the total overflow (tof). The fully utilized edges although do not contribute directly to the congestion, but they can contribute to blocking routing of the other nets. For any edge e_{ij}, the amount of overflow on it i.e., $overflow(e_{ij})$, and to determine if it is fully utilized i.e., $full(e_{ij})$ can be expressed using the following equations:

$$overflow(e_{ij}) = \begin{cases} u_{ij} - c_{ij} & \text{if } u_{ij} > c_{ij} \\ 0 & \text{otherwise}\,, \end{cases} \tag{1}$$

$$full(e_{ij}) = \begin{cases} 1 & \text{if } u_{ij} = c_{ij} \\ 0 & \text{otherwise}\,. \end{cases} \tag{2}$$

The tof is defined as equal to the tof of all edges and can be computed as follows:

$$tof(T) = \sum_{e_{ij} \in E} overflow(e_{ij})\,. \tag{3}$$

3. Proposed Global Router

3.1. *Overview*

The proposed global router executes three main tasks: (a) ordering of nets (b) initial routing of nets and (c) R&R process to eliminate congestion from the solution of initial routing. The tasks are distributed among several components. Figure 1 shows the UML component diagram[b] of the proposed global router that shows that it has three main components: (i) *Initial Routing*, (ii) *MRF method* and (iii) *R&R process*. These components work together to accomplish global routing. The component *Initial routing* executes the tasks of ordering and routing the nets. It uses the component *MRF method* to route the nets. The work of the *R&R process* component

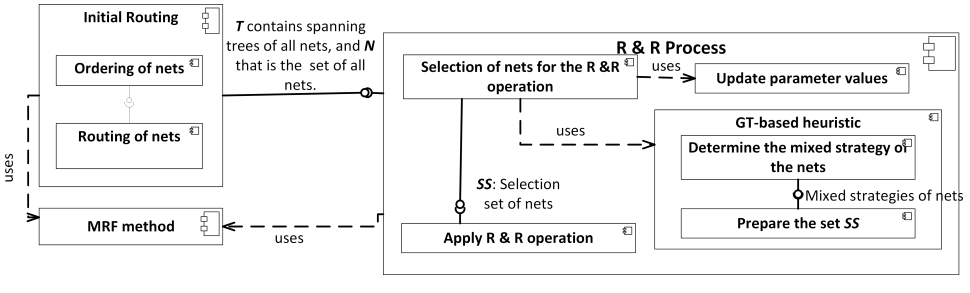

Fig. 1. Different components of the proposed global router.

[b] http://www.uml.org/ (accessed on 30 December, 2014).

starts after the completion of the initial routing phase and has several sub-components. The sub-component *Selection of nets for the R&R operation* uses the component *GT-based heuristic* to select nets. The component *GT-based heuristic* applies GT to determine the nets that should be ripped-up and rerouted in order to reduce the congestion of the solution. The nets selected for the R&R operation will be stored in the set SS and sent to the component *Apply R&R operation*. The component *Update parameter values* performs self-adjustment of parameters values based on the feedback of previous iterations in order to reduce the convergence time of the R&R process.

Figure 2 shows the different steps in the proposed global router. The first step is the reading of the 2D grid graph ($G(V, E)$), set of all nets (N) and parameters. The complete list of all parameters is described in Sec. 3.3.3, however, one parameter is described here which is $NUM\_THREADS$. $NUM\_THREADS$ sets the number of concurrent threads in the initial routing of nets. The second step is the sorting and initial routing of the nets. The nets are sorted in an ascending order of the area bounded by their pins. In this ordering, the nets whose pins are closer to each other are routed first. The rationale behind this ordering is that the nets whose pins are far from each other usually have more number of possible trees (i.e., alternate choices) available to them as compared to the nets whose pins are closer to each other. The nets that have more alternate choices available are less likely to be blocked from the routing of the other nets. The routing of nets can be done sequentially or concurrently. Most of the processors that are available these days have multi-cores and are capable of multithreading (MT). The routing of nets can be parallelized by using the following steps: (i) dividing the sorted nets in N among $\{N_0, N_1, \ldots, N_{NUM\_THREADS-1}\}$ subsets, (i) creating concurrent threads $tr_0, tr_1, \ldots, tr_{NUM\_THREADS-1}$ such that tr_i should route the nets in N_i (where $i = 0$ to $NUM\_THREADS$) and (iii) executing the threads in parallel. The threads should update the usage information of the edges after routing each net. The parallelization causes some increase in the congestion of the initial routing phase but that can be

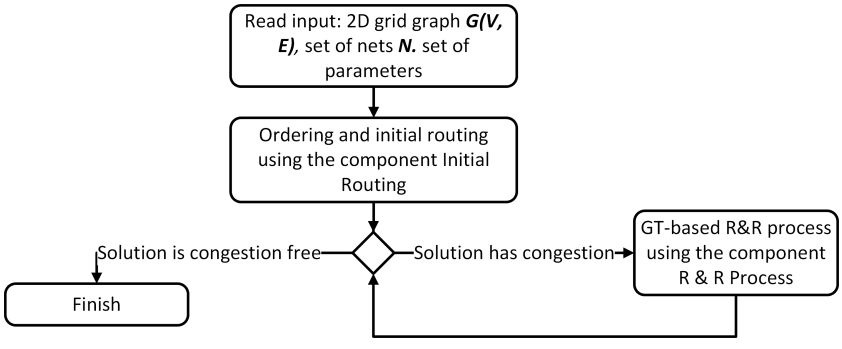

Fig. 2. Overall flow of the proposed global router.

compensated in the GT-based R&R process. After the completion of the initial routing phase, the R&R process is executed until the tof becomes zero.

3.2. *MRF method*

In MRF method, the routing region of each net is restricted to a rectangular region that covers its pins and some surrounding cells. Figure 3 shows the bounding region of a net which has three pins (p_0, p_1, p_2). The size of the bounding box can be increased and decreased using the parameter *BOX_SIZE*. Figure 4 shows the method of routing the nets. The variable t_i stores the spanning tree of net n_i. The *while* loop in its one iteration determines a branch to an uncovered pin, therefore, it usually has up to $m-1$ iterations (where m is the number of pins of n_i). The filling and retracing processes are similar to that of the Lee's algorithm on a weighted grid.[2] This work proposes a new function to determine the cost of any cell of the grid. For any cell $v_j \in V$ whose preceding cell in the filling process is v_i and the edge between them is represented as e_{ij}. The cost of v_j can be computed as follows:

$$cost(v_j) = 1 + e^{u_{ij}-(c_{ij}-\beta)} + cost(v_i)\,. \tag{4}$$

In Eq. (4), u_{ij} and c_{ij} represent the usage and capacity of the edge e_{ij} and the purpose of the exponential term is to avoid selection of paths through congested edges. When $(u_{ij} - (c_{ij} - \beta)) < 0$, the role of congestion is very limited and the algorithm find minimum length paths. However, when $(u_{ij} - (c_{ij} - \beta)) > 0$, then the role of

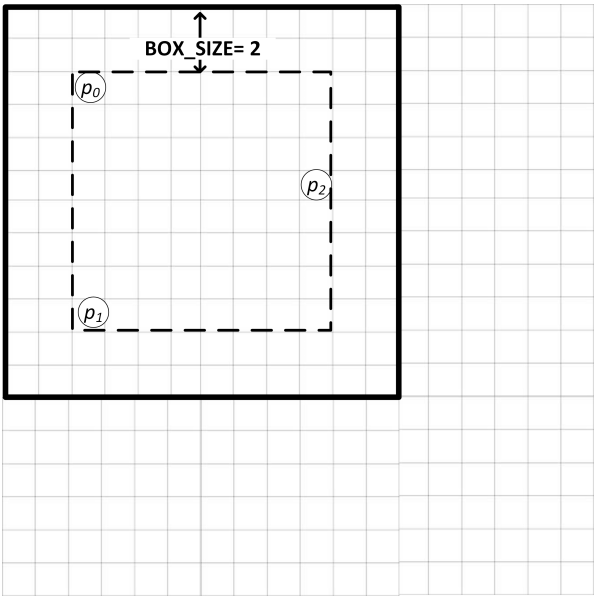

Fig. 3. Bounding region of MRF method for a net whose pins are (p_0, p_1, p_2).

Input: $P_i = \{p_0, p_1, ..., p_{m-1}\}$
Output: t_i: spanning tree of $n_i \in N$
1: Determine the bounded region
2: Randomly select a pin $p_x \in P_i$
3: **while** t_i is not completed **do**
4: Filling process from t_i to p_x, where $p_x \notin t_i, \in P_i$
5: Retrace process to form a branch from p_x to t_i and add the branch to t_i
6: **endwhile**

Fig. 4. Procedure of MRF method.

congestion costs becomes significant and the paths are determined w.r.t. minimum length as well as minimum congestion. The value of the parameter β is initially set by the user but can be varied in the R&R process in order to build spanning trees that have minimum length and/or minimum congestion. At the start of the filling process, the *cost* values of all cells in t_i are set to 0 and the *cost* values of the remaining cells in the bounded region is determined using the above mentioned equation. The filling process terminates when at least one of the following two conditions is reached: (a) the weights are assigned to all cells in the bounded region, or (b) a pin of the net which is not yet selected in t_i is found. The retracing process, forms a branch to a pin of the net to any one node (or cell) of t_i.

3.3. *R&R process*

The *R&R process* component executes the R&R process to eliminate congestion from the solution of initial routing. Figure 5 shows the steps in an iteration of the R&R process. The function of each component is described in the following.

3.3.1. *GT-based heuristic*

In this work, GT is used to solve the problem of deciding which nets should be ripped-up and rerouted in order to eliminate congestion. The selection of nets in the R&R process is modeled as a game in which nets act as players. The set N becomes the set of players of the game. Each net $n_i \in N$ has two pure strategies $S_i = \{Yes, No\}$. The strategy *Yes* means that n_i should attempt to improve its spanning tree and strategy *No* means that n_i should not attempt to improve its spanning tree. The players use mixed strategies to select their pure strategies. The mixed strategy of each player $n_i \in N$ is represented as $PR_i = (p_Y, 1 - p_Y)$, where p_Y is the probability of selecting the strategy *Yes* and $1 - p_Y$ is the probability of selecting the strategy *No*. The Nash equilibrium (NE)[12,13] of the game is reached when the tof of the solution becomes zero and at that point all nets want to stick to their *No* strategy (p_Y becomes zero for all nets). The aim of the proposed GT-based heuristic is to eliminate congestion, however, it could be extended to multiple objectives.

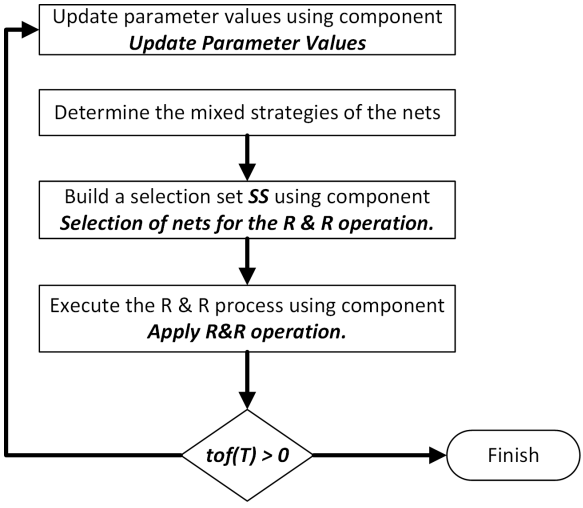

Fig. 5. Different steps in the R&R process.

The main idea behind the proposed method of computation of mixed strategies is as follows: Each player wants to achieve two goals: (i) Its spanning tree becomes congestion free and (ii) its spanning tree should not be blocking the routing of any other net. In any iteration, the nets which are more likely to progress towards achieving their goals will have higher values of mixed strategies and hence, are more likely to go through the R&R operation.

Figure 6 shows the method which computes the mixed strategies of nets. The input contains two parameters $CT1$ and TP. The parameter $CT1$ acts as the weight of the term related to the overflow of the spanning trees and TP is used to select the formula to compute the value of p_Y. The first formula of p_Y assigns higher values to the nets whose pins enclose a smaller area as compared to the other nets and p_Y of the nets which have no overflow is zero. The benefit of the first formula is that the nets that have congestion as well as require less time in rerouting should be preferred for the R&R operation as compared to others. The second formula of p_Y assigns values based on the following principles: (i) the nets that have more overflow and fully used edges will have a higher p_Y value, (ii) the weight of the overflow value in p_Y can be changed using the parameter $CT1$, and (iii) the nets that have not been ripped-up and rerouted since many iterations will have larger p_Y values. The p_Y values of the nets that have neither congested edges nor fully used edges is zero. The pseudo-code in Fig. 6 also shows that if none of the nets have any overflow then p_Y values of all nets becomes zero. When the value of parameter $CT1 \gg 1$, then the p_Y values rely more on the overflow of the edges as compared to other factors. When $CT \approx 1$, then p_Y values rely equally on overflow and number of fully used edges. When $CT1 \ll 1$, then p_Y values rely more on the number of fully used edges.

Input: T: Spanning trees of all nets, N: set of all nets, Parameters: $CT1 \in Z$, $TP \in \{True, False\}$
Output: $PR = \{PR_0, PR_1, ..., PR_{N-1}\}$: mixed strategies of all nets

for each net $n_i \in N$ **do**
  $ofl(n_i) = \sum_{e_i \in t_i} overflow(e_i)$
  $fl(n_i) = \sum_{e_i \in t_i} full(e_i)$
  $area(n_i)$= area bounded by the pins (p_i) of n_i.
  $Iter(n_i)$= Number of iterations since n_i was ripped-up and re-routed last time.
endfor
if $\sum_{n_i \in N} ofl(n_i) > 0$ **then**
  Normalize values in ofl, fl and $Iter$.
  for each net $n_i \in N$ **do**

$$p_Y(i) = \begin{cases} \frac{1}{area(n_i)} \text{ if } TP = 0,\ ofl(n_i) > 0, \\ \frac{CT1 \times ofl(n_i) + fl(n_i)}{m} \times (1 + Iter(n_i)) \text{ if } TP = 1, \\ 0 \text{ otherwise} \end{cases}$$

  endfor
  Normalize p_Y values
else
  for i=0 to $N-1$ **do**
    $p_Y(i) = 0$
  endfor
endif
for i=0 to $N-1$ **do**
  $PR_i = \{p_Y(i), 1 - p_Y(i)\}$
endfor
return (PR)

Fig. 6. Method to determine the mixed strategies of all nets in N.

The next step is to prepare a set SS that contains the nets whose spanning trees should be rip-up and rerouted. In the selection of nets in SS, the p_Y values of the nets act as their probabilities with which they could be selected in SS.

3.3.2. *Apply R&R operation*

The input of the component *Apply R&R operation* is the set SS. This work uses two types of R&R operations: (i) R&R Type A and (ii) R&R Type B. The Type A operation rips-up and reroutes one net at time, whereas, the Type B operation first rips-up two nets and then reroutes them. In both type of operations, the nets are ripped-up and rerouted completely. The R&R process is executed sequentially. The nets in SS are divided into two subsets SS_A and SS_B such that SS_B contains $RR1\%$ nets of SS and $SS_A = SS - SS_B$. Furthermore, the number of nets in SS_B should be even, otherwise, one element should be moved from SS_B to SS_A. The nets in SS_A go though the Type A operation and the nets in SS_B go through the Type B operation. The proposed global router uses two types of R&R operations because it was

observed by the authors in their experiments that convergence time can be reduced by using more than one type of R&R operation.

The R&R process is executed as follows: A net (n_i) is fetched from SS, if $n_i \in SS_A$, then Type A operation is applied to it. If $n_i \in SS_B$, then another net $n_j \in SS_B$ is fetched from SS and Type B operation is applied to the nets n_i and n_j. The main steps in both types of R&R operations are as follows: (i) copy the spanning tree(s) of the selected nets into temporary variables, (ii) completely delete (rip-up) the existing spanning tree(s), (ii) create new tree(s) using MRF method and (iv) compare the new tree(s) with the existing ones (i.e., the trees stored in temporary variables) and keep the better one(s). The comparison is required to ensure that the R&R process is moving towards congestion elimination with limited amount of hill-climbing. Figure 7 shows the function that compares two spanning trees t_i and t_i^* of a net n_i, where t_i^* is the new spanning tree and t_i is the original spanning tree. The function returns *true*, if t_i^* is better than t_i. In the comparison function, if the inferior solutions are always rejected then there is a high chance that our R&R process can get stuck into a local optima because of no hill-climbing. Therefore, the R&R operation occasionally allows acceptance of inferior solutions whose difference in the tof value with the original solution is not more than $-NEG$, where NEG is a parameter which can be set by the user.

Input: $G(V, E)$, t_i, t_i^*: two spanning trees for a net n_i, $NEG \in Z^+$
Output: $Y \in true, false$

1: $Y = false$
2: $A = \sum_{e_x \in t_i} overflow(e_x)$
3: $B = \sum_{e_x \in t_i} full(e_x)$
4: $C = |t_i|$
5: $D = \sum_{e_x \in t_i^*} overflow(e_x)$
6: $E = \sum_{e_x \in t_i^*} full(e_x)$
7: $F = |t_i^*|$
8: r_{NEG}= a random integer between $-NEG$ and 0
9: **if** $D < A$ or $(D - A) \leq r_{NEG}$ **then**
10: $Y = true$
11: **else**
12: **if** $D == A$ and $E < B$ **then**
13: $Y = true$
14: **else**
15: **if** $D == A$ and $B == E$ and $F < C$ **then**
16: $Y = true$
17: **endif**
18: **endif**
19: **endif**
20: **return** Y

Fig. 7. Method of comparing two trees (t_i, t_i^*) of net n_i.

3.3.3. *Update parameter values*

The proposed global router has two types of parameters: static parameters and self-adjustable (adaptable) parameters. The values of static parameters are set by the user and remains constant throughout the execution. The values of self-adjustable parameters change following an arithmetic progressions (APs) in the component *Update parameter values*. The AP of any parameter can be completely specified by three terms: (first term, last term and difference). The first term in its initial value, the last term is its maximum value and the difference is the amount by which it increments during self-adjustment. The user specifies the three terms of AP for each self-adjustable parameter. During the execution of global router, the component *Update parameter values* update the parameter value using their APs. In most of the self-adjusting parameters, when their value becomes equal to the last term of their AP, then their next value is the first term of their AP. Table 1 shows all parameters of the proposed global router and classifies them as static or self-adjustable. The parameter T_m represents a threshold value for tof value of the current iteration such that if its value becomes smaller than T_m, then the value of BOX_SIZE is set equal to its last term. The parameter $CT2$ represents the number of preceding iterations whose tof values are used in adjusting the parameters values.

Figure 8 shows the procedure that is applied in each iteration of the R&R process in the component *Update parameter values*. The inputs are: Current iteration count and tof values of the current iteration and that of last $CT2$ iterations. In the first iteration (i.e., $i = 0$), the values of parameters are initialized and later on, the values of the parameters are adjusted based on overflow value of its preceding iterations. In the first iteration, the value of $TP1$ is assigned to zero, however, it changes to one based on the conditions mentioned in Fig. 8. The benefit of self-adjusting parameters is that they can help in exploring unique spanning trees for the nets. The proposed global router employs only MRF method of routing, and if a same method is applied to a net multiple times with same parameters values, then there is a high possibility that it returns a same solution every time. However, the chances of getting a different spanning tree for a net increases significantly by using different parameter values.

Table 1. List of all parameters.

Parameters	Type	Reference	AP representation
RR1	Self-adjustable	Section 3.3.2	$RR1(RR1_i RR_f, RR_d)$
CT1	Self-adjustable	Section 3.3.1	$CT1(CT1_i, CT1_f, CT1_f)$
TP1	One-time self-adjustable	Section 3.3.1	—
β	Self-adjustable	Section 3.2	$\beta(\beta_i, \beta_f, \beta_d)$
BOX_SIZE	Self-adjustable	Section 3.2	$BOX\_SIZE(BOX\_SIZE_i, BOX\_SIZE_f, BOX\_SIZE_d)$
CT2	Static	Section 3.3.3	—
NEG	Static	Section 3.3.2	—
T_m	Static	Section 3.3.3	—

Input: i: current iteration, Overflow values of last $CT2$ iterations: $\{tof(T_i)$, $tof(T_{i-1}$, ..., $tof(T_{i-CT2})\}$

1: **if** i=0 **then**
2: $RR1 = RR1_i, CT1 = CT1_i, BOX\_SIZE = BOX\_SIZE_i, \beta = \beta_i$
3: $TP1 = 0$
4: **else**
5: $RR1 = RR1 + RR1_d$
6: **if** $RR1 > RR1_f$ **then**
7: $RR1 = RR1_i$
8: **endif**
9: **if** $tof(T_i) = tof(T_{i-1})$ **then**
10: $CT1 = CT1 + CT1_d$
11: $TP1 = 1$
12: **endif**
13: **if** $CT1 > CT1_f$ **then**
14: $CT1 = CT1_i$
15: **endif**
16: **if** $tof(T_i) < T_m$ **then**
17: $BOX\_SIZE = BOX\_SIZE_f$
18: **else**
19: **if** $tof(T_i) = tof(T_{i-1}) = ... = tof(T_{i-CT2})$ **then**
20: $BOX\_SIZE = BOX\_SIZE + BOX\_SIZE_d$
21: **if** $BOX\_SIZE > BOX\_SIZE_f$ **then**
22: $BOX\_SIZE = BOX\_SIZE_f$
23: **endif**
24: $\beta = \beta + \beta_d$
25: **if** $\beta > \beta_f$ **then**
26: $\beta = \beta_i$
27: **endif**
28: **endif**
29: **endif**
30: **endif**

Fig. 8. Procedure to update the parameter values.

4. Experimental Results

The proposed global router was implemented using C++ and compiled using the Intel compiler for Linux. The code was executed on a virtual machine (VM) of the university's cloud computing facilities.[c] The VM has eight 2 GHz virtual processors, 16 GB of memory and RedHat Linux OS. It can execute up to eight concurrent threads. The proposed global router was implemented with the following parameter values: $RR1 = (10, 60, 5)$, $CT1 = (1, 10, 1)$, $BOX\_SIZE = (1, \max(\frac{X_{\max}}{2}, \frac{Y_{\max}}{2}), 5)$ (where $X_{\max}$ and $Y_{\max}$ represent the horizontal and vertical grid sizes), $\beta = (1, 5, 1)$,

[c]https://cloud.kfupm.edu.sa/ (accessed on 30 December, 2014).

$CT2 = 2$, $NEG = 2$, $T_m = 2$. The value of $TP1$ do not need to be set as it is set internally by the component *Update parameter values*. The implementations of the proposed global router are represented as *Proposed*(x), x is the value of parameter *NUM_THREADS*. In the experiments, x value was kept equal to 1, 4 and 8, because modern processors can usually execute four (dual-core) or eight (quad-core) concurrent threads.

The parameters were set based on the experimental observations to optimize both speed and solution quality. The value of $RR1$ lies between 10 and 60, so that both types of R&R operations can be applied to the nets. The parameter $CT1$ is used by the formula which computes the mixed strategies of nets. The mixed-strategy formula of each net contains two terms, the first one is the overflow of its spanning tree and the second one is the number of fully used edges in its spanning tree. The parameter $CT1$ is the weight of the overflow term. When the value of $CT1$ is large, then the nets which have large overflow also have large mixed strategies than the nets which have small overflow but large number of fully used edges. When the value of $CT1$ is small, then the nets which have a small overflow but large of number of fully-used edges also have large mixed strategies. If the value of $CT1$ is too large, then nets having no-zero overflow will be only selected in the R&R operation. The value of $CT1$ is varied so that nets having spanning trees of different characteristics can go through the R&R operation. The value of *BOX_SIZE* was initially set to one, so as to speed-up the initial routing phase. The *NEG* value was set so as to allow small amount to hill-climbing. The value of T_m is set to a small value because when the R&R process is near completion, then a small number of nets participate in the R&R process which should be routed using maximum resources. It was also noticed that a different set of parameter values can produce solutions that are quite different in terms of wire-length and execution time. The results and binaries of our global router for the Linux 64-bit platform are available at the author's website.[d]

The test problems of the ISPD'98 suite[17,a] were used to evaluate the performance of the proposed global router. Table 2 shows the characteristics of the problems in the ISPD'98 suite. Almost all heuristics use random number generators and therefore

Table 2. The ISPD'98 test problems.

Test problem	# of nets	Grid size	Test problem	# of nets	Grid size
ibm01	11507	64 × 64	ibm02	18429	80 × 64
ibm03	21621	80 × 64	ibm04	26163	96 × 64
ibm05	27777	128 × 64	ibm06	33354	128 × 64
ibm07	44394	192 × 64	ibm08	47944	192 × 64
ibm09	50393	256 × 64	ibm10	64227	256 × 64

[d]https://sites.google.com/site/ufsresearch/ (posted on 30 December 2014).

their performance may become dependent upon the sequence of the random number generated by the generator. The sequence is controlled by the seed value of the generator. Therefore, the effect of different seed values should be considered in the experiments. The results of the proposed global router are not dependent on any particular seed value. In the experiments, each problem was solved for up to hundred times (i.e, have 100 trials) with a unique seed value. The proposed global router was compared with two recent routers: Box Router 2.0[19] and Side-winder.[20,21] The executable of Box Router 2.0 was obtained from its authors and executed on the same VM on which we executed our proposed global router. The parameters of Box Router 2.0 were set so as to optimize it for speed and obtain a valid solution in smallest possible time. The parameters values were set as follows: MULTI_LAYER = 0, PREROUTING = 1, BOXROUTING = 1, BOXROUTING_STEP = 1600, BOXROUTING_ILP = 0, ILP_SOLVER = 1, REROUTING = 1, REROUTING_COUNT = 1000, REROUTING_REPEAT = 0, REROUTING_STEP = 100, REROUTING_PUSH = 3, REROUTING_STUCK = 5, MAZEROUTING_MARGIN = 150, and RELAYERING = 0. Box Router 2.0 uses GNU Linear Programming Kit (GLPK)[e] as an external ILP solver. It employs GLPK using its C++ application programming interfaces (APIs) which do not support MT. Therefore, Box Router 2.0 executed as a single-threaded program on the multi-core VM used in the experiments. The source code or executable of Side-winder is not publicly available, therefore, its published results[20,21] were used in comparison.

In the global routing problem, the primary objective is to find a valid or congestion-free solution and the secondary objectives include minimization of wirelength and execution time. In our experiments, the proposed global router and Box Router 2.0 has solved all ten problems of the ISPD'98 suite. The Side-winder has

Table 3. Tof of the solutions obtained from different global routers.

	Tof		
Problem	*Proposed*$(x), x \in \{1, 4, 8\}$	Box Router 2[19]	Side-winder[20,21]
ibm01	0	0	255
ibm02	0	0	8
ibm03	0	0	0
ibm04	0	0	618
ibm05	0	0	0
ibm06	0	0	0
ibm07	0	0	0
ibm08	0	0	0
ibm09	0	0	0
ibm10	0	0	0

[e]https://www.gnu.org/software/glpk/ (accessed on 30 December, 2014).

solved up to seven problems. Therefore, the proposed global router is better than Side-winder in the most important goal of global routing which is routability.

Tables 4 and 5 report the results of the wire-length and runtime, respectively. The results of *Proposed*(8) consists of three columns that are represented as median, Q1 and Q3. The medium column contains the medium of all the trials. The columns Q1 and Q3 contains the lower quartile or 25th percentile and upper quartile or 75th percentile of the results in different trials, respectively. Q1 and Q3 indicate the minimum and maximum values between which results of most of the trials exist. The results of *Proposed*(4) and *Proposed*(1) consist of only one column which contains the median of the trials. The last column has three or four sub-divisions and contains the gain which is expressed as the difference between the median result of the proposed router and the result of the Box Router 2.0 or Side-winder.

Table 4 shows that the solutions of the proposed global router (*Proposed*(x), for $x = 1, 4$ or 8) have wire-lengths better than that of Box Router 2.0 in up to seven test problems (i.e., ibm02, ibm03 and ibm06-to-ibm10). The Side-winder has obtained better wire-lengths in many test problems, however, it could not solve three test problems.

Table 5 shows that median as well as upper quartile of the trials of *Proposed*(8) and *Proposed*(4) are better than that of Box Router 2.0 in up to eight test problems (ibm02, ibm03 and ibm05-to-ibm10). In six test problems (ibm01-to-ibm03, ibm07, ibm09 and ibm10) the runtime of *Proposed*(1) is better than or at most 2.6 s more than that of Box Router 2.0.

Tables 4 and 5 conveys the following information about the comparison between the proposed global router (*Proposed*(8) or (*Proposed*(4))) and Box Router 2.0. In seven problems in which the proposed global router found solutions with smaller wire-length, it also found with better runtime. The parameters of Box router 2.0 were adjusted for speed, therefore, the results show that the proposed global router is faster than Box Router 2.0 in finding valid solutions. A different parameter values can be set in Box Router 2.0 which could improve its wire-length at the expense of increase in runtime. The Box Router 2.0 does not use MT because it employs an external ILP solver GLPK that does not support MT. The use of a faster ILP solver is a topic of future research for Box Router 2.0.[18,19]

Figure 9 shows the results of peak memory usage of the proposed global router. The results shows that the memory usage of the proposed global router depends directly on the wire-length and number of concurrent threads. Memory usage of the proposed global router with NUM_THREADS = 8 remains below 1.5 GB.

In addition to GT, the proposed global router also uses MRF and MT techniques to quickly converge to a valid solution. The proposed GT-heuristic has a key role in minimizing the runtime. Figure 10 shows that the absence of the proposed GT-based heuristic from the proposed global router causes an increase in its runtime in all problems.

Table 4. Wire-lengths of the proposed global router, Box Router 2.0[19] and Side-winder.[20,21]

	Proposed(8)			*Proposed*(4)	*Proposed*(1)	Box Router 2.0	Side-winder	Gain			
Problem	Median (x_1)	Q1	Q3	Median (x_2)	Median (x_3)	(x_4)	(x_5)	($x_4 - x_1$)	($x_4 - x_2$)	($x_4 - x_3$)	($x_5 - x_1$)
ibm01	66953.00	66717.00	67271.00	66875.00	66801.00	66146.00	—	−807.15	−729.00	−655.00	—
ibm02	1755164.50	174921.50	175683.00	175240.00	175388.00	179247.00	—	4082.50	4007.00	3859.00	—
ibm03	149076.00	149029.50	149182.00	149116.00	149054.50	152029.00	147524.00	2953.00	2913.00	2974.50	−1552.00
ibm04	174419.00	174133.50	174601.00	174224.00	174333.00	172707.00	—	−1712.00	−1517.00	−1626.00	—
ibm05	417756.50	417703.50	417792.00	417738.50	417730.00	416913.00	409778.00	−843.50	−825.50	−817.50	−7978.50
ibm06	286025.00	285659.00	286448.00	285683.00	285849.00	287302.00	280007.00	1277.00	1619.00	1453.00	−6018.00
ibm07	373977.00	373737.50	374325.00	374057.00	373969.50	378789.00	381694.00	4812.00	4732.00	4819.50	7717.00
ibm08	414076.50	413761.50	414457.00	413882.50	413684.50	417449.00	413300.00	3372.50	3566.50	3522.00	−776.50
ibm09	421702.50	421544.00	422112.00	421721.50	421484.50	431579.00	416554.00	9876.50	9857.50	9929.00	−5148.50
ibm10	593544.50	593095.00	594020.00	593404.00	592825.00	605348.00	591036.00	11803.50	11944.00	11937.00	−2508.50
Total gain $\sum$								34814.50	35567.50	35395.00	−16265.00

Table 5. Runtime of the proposed global router and Box Router 2.0.[19]

	Proposed(8) (s)			*Proposed*(4) (s)	*Proposed*(1) (s)	Box Router 2.0 (s)	Gain (s)		
Problem	Median (x_1)	Q1	Q3	Median (x_2)	Median (x_3)	(x_4)	($x_4 - x_1$)	($x_4 - x_2$)	($x_4 - x_3$)
ibm01	5.55	4.93	6.39	4.60	5.25	4.95	−0.60	0.35	−0.30
ibm02	4.60	4.92	5.06	4.66	8.29	7.20	2.60	2.54	−1.10
ibm03	2.32	2.05	2.60	2.57	5.04	6.47	4.15	3.90	1.43
ibm04	62.08	51.50	72.81	58.72	57.43	39.81	−22.27	−18.91	−17.62
ibm05	2.46	2.39	2.71	4.53	17.10	10.34	7.88	5.81	−6.76
ibm06	8.87	8.25	9.74	9.85	18.65	11.17	2.30	1.32	−7.49
ibm07	7.70	6.98	8.70	8.29	15.83	17.67	18.85	18.26	10.72
ibm08	11.83	11.34	12.78	14.59	30.56	20.91	9.08	6.32	−9.66
ibm09	8.80	8.15	9.77	10.17	20.39	19.51	10.71	9.34	−0.88
ibm10	15.45	13.26	17.11	16.81	33.12	30.58	15.13	13.77	−2.54
Total gain $\sum$							47.83 s	42.69 s	−34.21 s

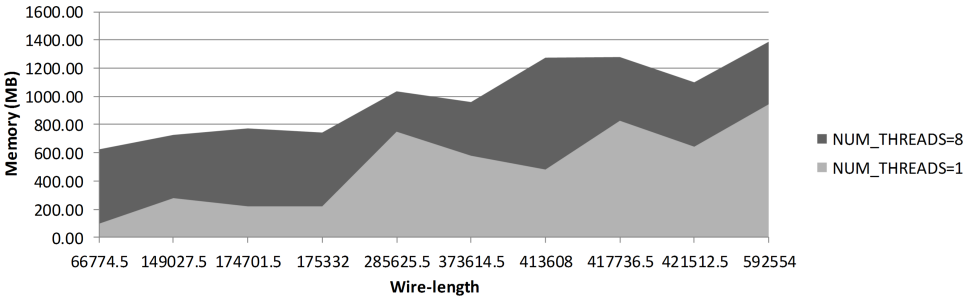

Fig. 9. Memory usage of the proposed global router.

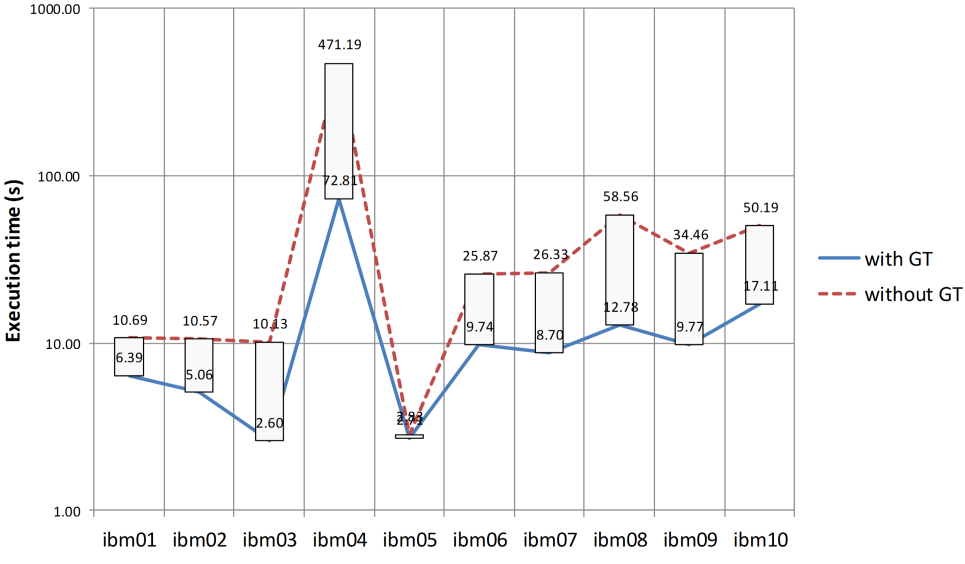

Fig. 10. Increase is runtime after removing GT-based selection of nets from the proposed global router.

5. Conclusion

In this paper we presented a GT-based global router that employs MRF method and GT-based R&R process. In the GT-based R&R process, the nets act as players and each net has two pure strategies: (i) Attempt to improve its spanning tree and (ii) do not attempt to improve its spanning tree. The nets use mixed strategies to select any particular pure strategy. A method is proposed for the determination of mixed strategies. The nets that have selected to improve their spanning trees will go through the R&R operation. The proposed global router was implemented using C++ and the benchmarks of the ISPD'98 suite were used in performance evaluation. The proposed global router was compared with two recent ILP-based global routers,

which are: (i) Box Router 2.0 and (ii) Side-winder. The proposed global router with MT is faster than Box Router 2.0. The proposed global router has solved more problems as compared to Side-winder, therefore, it is better than Side-winder in terms of routability. The experimental results show that the proposed GT-based heuristic is effective in reducing the runtime of a simple maze router.

Acknowledgments

The authors wish to acknowledge King Fahd University of Petroleum & Minerals, Dhahran, Saudi Arabia for all the support.

References

1. T. Langauer, *Combinatorial Algorithms for Integrated Circuit and Layout* (John Wiley & Sons, New York, 1990).
2. Sadiq M. Sait and Habib Youssef, *VLSI Physical Design and Automation: Theory and Practice* (World Scientific Publishers, Singapore, 1999).
3. H.-Y. Chen and Y.-W. Chang, *Electronic Design Automation: Synthesis, Verification, and Testing*, L. T. Wang, Y.-W. Chang and K.-T. Cheng (eds.) (Elsevier Morgan Kaufmann, Burlington, USA, 2009), pp. 687–749.
4. X. He, T. Huang, L. Xiao, H. Tian and E. F. Y. Young, Ripple: A robust and effective routability-driven placer, *IEEE Trans. Computer-Aided Design of Integr. Circuits Syst.* **32** (2013) 1546–1556.
5. M. Pan and C. Chu, Fast route: A step to integrate global routing into placement, *IEEE/ACM Int. Conf. Computer-Aided Design*, San Jose, CA, November 2006, pp. 464–471.
6. K. Suzuki, Y. Matsunaga, M. Tachibana and T. Ohtsuki, A hardware maze router with application to interactive rip-up and reroute, *IEEE Trans. Computer-Aided Design* **CAD-5** (1986) 466–476.
7. M. Pan, Y. Xu, X. Zhang and C. Chu, FastRoute: An efficient and high-quality global router, *VLSI Design* (Hindawi Publishing Corporation, New York, USA, 2012).
8. Yiding Han, Dean Michael Ancajas, Koushik Chakraborty and Sanghamitra Roy, Exploring high-throughput computing paradigm for global routing, *IEEE Trans. VLSI Systems*, **22** (2014) 155–167.
9. M. Elghazali, S. Areibi, G. Grewal, A. Erb and J. Spenceley, A comparison of hardware acceleration methods for VLSI maze routing, *IEEE Toronto Int. Conf. Science and Technology for Humanity (TIC-STH)*, Toronto, Canada, September 2009, pp. 563–568.
10. Y. Han, D. M. Ancajas, K. Chakraborty and S. Roy, Exploring high-throughput computing paradigm for global routing, *IEEE Trans. VLSI Syst.* **22** (2014) 155–167.
11. T.-H. Wu, A parallel integer programming approach to global routing, Ph.D. Thesis, University of Wisconsin-Madison (2011).
12. E. N. Barron, *Game Theory: An Introduction*, 2nd edn. (John Wiley & Sons, Hoboken, USA, 2013).
13. N. Nisan, T. Roughgarden, E. Tardos and V. V. Vazirani, *Algorithmic Game Theory* (Cambridge University Press, New York, USA, 2007).
14. F.-N. Pavlidou and G. Koltsidas, Game theory for routing modeling in communication networks — A survey, *J. Commun. Networks* **10** (2008) 268–286.

15. A. Salhi and Ö. Töreyen, *Computational Intelligence in Optimization* (Springer-Verlag Berlin, Heidelberg, 2010), pp. 211–232.
16. L. A. DaSilva, H. Bogucka and A. B. MacKenzie, Game theory in wireless networks, *IEEE Commun. Magaz.* **49** (2011) 110–111.
17. C. J. Alpert, The ISPD98 Circuit Benchmark Suite, *Int'l Symp. Physical Design (ISPD)*, Monterey, CA (1998).
18. M. Cho and D. Z. Pan, BoxRouter: A new global router based on box expansion and progressive ILP, *IEEE Trans. Computer-Aided Design Integr. Circuits Syst.* **26** (2007) 2130–2143.
19. M. Cho, K. Lu, K. Yuan and D. Z. Pan, BoxRouter 2.0: A hybrid and robust global router with layer assignment for routability, *ACM Trans. Design Automation of Electronic Systems* **14** (2009) 32.
20. J. Hu, J. A. Roy and I. L. Markov, Sidewinder — A scalable ILP-based Router, *Proc. Int. Workshop on System Level Interconnect Prediction (SLIP)* (ACM Press, 2008), pp. 73–80.
21. J. Hu, High-performance global routing for trillion-gate systems-on-chips, Ph.D. Thesis, Computer Science and Engineering, University of Michigan (2013).