CHAPTER 5

SPEEDUP TECHNIQUES FOR DYNAMIC NETWORKS USING RSB PARTITIONING STRATEGY

5.1 PREAMBLE

All the existing research techniques on speedup techniques to the shortest path queries concentrate on time-independent shortest path (static) routing. At present, research is focused on time dependent dynamic shortest path routing. Dynamic shortest path routing with time dependent network on real world systems has been addressed in [28 – 45]. Many researchers have addressed speedup techniques using dynamic arc-flags in real world applications [46-49]. Combining speedup techniques are also applied in dynamic shortest path routing problems [59-64].

The speedup techniques compute additional data during the pre-processing phase to increase the speed of execution of queries during online processing. But not all of those additional data are proved to yield the expected result in situations where the edge weights will change due to the traffic jams or the delay of trains. Such situations arise frequently in practice. In many cases, variants of the arc-flag method [28] are used in these scenarios. The partitioning strategy adapted in the arc-flag method plays an important role to accelerate the query responses. The rectangular/grid partitioning considered in arc-flag approach has been always compared with other partitioning strategies [28, 67]. Therefore, arc-flag method using grid partitioning is taken as a baseline and it is compared with the proposed RSB partitioning-based speedup techniques. Figure 5.1 presents the proposed partitioning-based speedup techniques for dynamic networks.
In this research, first, a new partitioning technique for the arc-flag speedup technique based on the Recursive Spectral Bisection method (Arc-RSB) is proposed. Second, the Bidirectional search and RSB (Bi-RSB) partitioning strategy is also proposed. The above proposed techniques are parallelized to improve the speed of the multi-core system. Subsequently, Para-Arc-RSB and Para-Bi-RSB are the new proposed parallelized speedup techniques. For all these techniques, parameters such as pre-processing time, update time and runtime are calculated and the metric QPL is used to measure the performance of the system.

5.2 MODELING DYNAMIC SHORTEST PATH PROBLEM

5.2.1 Overview

The dynamic shortest path problem is formulated as a minimization problem for standard Shortest Path Problem (SPP) at time ‘t’. The dynamic network deals with graphs where the cost function changes or is updated after a certain time interval, but the graph is static between two consecutive cost function changes. This variant is called as cost-update. The corresponding SPP on dynamic graphs/dynamic networks is called Time-Dependent Cost Updates Shortest Path Problem (TDCUSPP) [46].
5.2.2 Time-Dependent Cost Updates Shortest Path Problem (TDCUSPP)

A graph is said to be dynamic when some of the graph entities (vertices, edges and weights) change with time. The most usual time-dependent changes are in the edge weights which can also model edge connection/disconnection. This type of network can be modeled as a directed graph $G = (V,E(t))$, where $V = \{v_1, v_2, \ldots, v_n\}$ is a set of ‘n’ vertices and $E = \{e_1(t), e_2(t), \ldots, e_m(t)\}$ is a set of ‘m’ edges at time ‘t’. For the dynamic case, an $(s, d)$-path $P$ in graph $G=(V, E(t))$ at time ‘t’ is called as $(s, d, t)$-path and is denoted by $(P, t)$. The shortest path length $l(P, t)$ of a $(s, d)$-path is defined by,

$$l(P, t) = \sum_{i=1}^{k-1} l_{v_i v_{i+1}}(t)$$

where $l_{v_i v_{i+1}}(t)$ is the length of the path connecting vertex ‘$v_i$’ and ‘$v_{i+1}$’ at time ‘t’. A $(s, d, t)$-path $(P, t)$ that minimizes the shortest path length $l(P, t)$ is called as $(s, d, t)$-shortest path at time ‘t’ and the length is denoted by $\text{dist}(s, d)$. It is also required to execute the speedup techniques in dynamic scenarios so that the efficiency of query processing will be improved.

In this section, the speedup techniques, namely, the Arc-flag method, the Recursive Spectral Bisection method (RSB) and the Bidirectional search are modeled for dynamic scenarios. TDCUSPP is modeled for dynamic graphs using two threads. They are flag thread and time thread. Flag thread is used to determine change in edge weight. Timer thread is used to change the edge weight in random time for making the graph dynamic. Also, the usage of two threads using parallel programming techniques is explained in this section.

Arc-Flags in Dynamic Scenarios

Given a dynamic graph $G = (V,E(t))$, where $V = \{v_1, v_2, \ldots, v_n\}$ is a set of ‘n’ vertices and $E = \{e_1(t), e_2(t), \ldots, e_m(t)\}$ is a set of $m$ edges at time ‘t’ and a partition $R = \{R_1, R_2, \ldots, R_r\}$ of $V$. The shortest path problem for the arc-flag speedup technique consists of an arc-flag vector, which is a single dimensional array whose size is
equivalent to the number of grids $R$ (number of regions) in rectangular partition. The shortest path is the summation of sub-paths obtained in each region $r \in R$. The search ends in each region in its boundary vertex ‘$b$’ for which the arc-flag vector is set to true i.e., $a(r) = 1$. Hence, the modified shortest path can be given as,

$$l(P,t) = d_u + \min \sum_{i=u}^{b-1} a_r \cdot l_{i,i+1}(t) \quad r \in R \text{ and } 1 \leq r \leq R \quad (5.2)$$

where ‘$a_r$’ is set to true when $l_{i,i+1}(t)$ is part of shortest path and ‘$r$’ is a shortest path region. Otherwise, ‘$a_r$’ is false. Here, ‘$d_u$’ is the shortest path distance from source ‘$s$’ to vertex ‘$u$’.

The easiest way to partition a graph with a 2D layout is to define the regions with a $n \times m$ grid of the bounding box [28]. The top-left coordinate of the bounding box of the 2D layout of the graph is denoted with $(l, t)$ and bottom-right one is denoted with $(r, b)$. Furthermore, $w = r - l$ is defined as the width and $h = t - b$ is the height of the layout. The grid cell or region $G_{i,j}$ with $0 \leq i < n, 0 \leq j < m$ is now defined as the rectangle.

$$\left[ l + i \frac{w}{n}; l + (i + 1) \frac{w}{n} \right] \star \left[ b + j \frac{h}{m}; b + (j + 1) \frac{h}{m} \right] \quad (5.3)$$

Nodes on a grid line are assigned to one of the neighboring grid cells. Figure 5.2 below shows an example of a $6 \times 6$ grid.

![Figure 5.2 Grid Partitioning of size 6 x 6](image-url)
The rectangular or grid partitioning method uses only the bounding box of the graph and all other properties like the structure of the graph or the density of nodes are ignored.

**Recursive Spectral Bisection (RSB)**

The experimental evaluation for time-dependent arc-flags [48, 49] on real world road networks shows that the decrease in query performance is minimal compared to the speed-up gained in the update phase. But it is required to improve the query performance which decides the efficiency of the system. Hence, a new domain specific natural partitioning called Recursive Spectral Bisection (RSB) is applied in speedup techniques which will improve the efficiency of query processing. As RSB adopts a domain specific partitioning, there is no need for any data structure which has to be updated at regular intervals of time. The RSB partitioning procedure is explained in Figure 5.3.

**Steps:**

1. Construct Laplacian Matrix for the given Graph G.
2. Find the second Eigenvector of the Laplacian Matrix (Fiedler Vector)
3. Sort the Fiedler Vector in order
4. Bisect the Graph using the Sorted Fiedler Vector

**Figure 5.3 RSB Partitioning**

The RSB algorithm is derived from a graph bisection strategy developed by Pothen, Simon, and Liou. The RSB partitioning is based on the computation of a specific eigenvector of the Laplacian matrix of the graph G. The Laplacian matrix $L(G) = (l_{ij})$, $i, j = 1…n$ is defined by,
From the definition of $L$, it also follows that the largest eigenvalue $\lambda_1$ is zero, and that the associated eigenvector is $\bar{e}$, the vector of all ones. This is simply a consequence of the particular choice of diagonal elements in $L(G)$. If $G$ is connected, then $\lambda_2$ the second largest eigenvalue is negative. The magnitude of $\lambda_2$ is a measure of connectivity of the graph or its expansion.

The Lanczos algorithm is an iterative algorithm invented by Cornelius Lanczos and is an adaptation of power methods to find eigen values and eigen vectors of a square matrix or the singular value decomposition of a rectangular matrix. It is mostly useful for finding decompositions of very large sparse matrices. Here, the Lanczos algorithm is used, since it does not require any manipulation of the Laplacian matrix $L(G)$. In addition, matrix vector multiplications with $L(G)$ are also needed. This multiplication can be implemented at no additional storage cost, since the Laplacian matrix directly reflects the structure of the graph.

**Bidirectional Search in Dynamic Scenario**

Bidirectional search simultaneously applies the forward variant of the algorithm beginning at the source node and a so-called reverse or backward variant of Dijkstra’s algorithm starting at the destination node. With the reverse variant, the algorithm is applied to the reverse graph, that is, a graph with the same node set $V$ as that of the original graph, and the reverse edge set $E = \{(u, v) \mid (v, u) \in E\}$. The distance labels of the forward search is assumed as $df(u)$ and the labels of the backward search is assumed as $db(u)$. The algorithm can be terminated when one node has been designated to be permanent by both the forward and the reverse algorithms. The shortest path is then determined by node ‘$u$’ with minimum value $df(u) + db(u)$ and can be composed of the
one from the start node to ‘$u$’ found by the forward search and the edges reverted again on the path from the destination to ‘$u$’, found by the reverse search.

The bidirectional search consists of two graphs, namely, forward graph $G_f(V_f,E_f)$ and reverse graph $G_r(V_r,E_r)$. Hence, the bidirectional shortest path is the sum of sub-paths obtained from forward graph from source and reverse graph from destination at time ‘$t$’ and is given by,

$$l(P,t)= P_f(t)+P_r(t)$$

where $P_f(t)$ is the shortest sub-path obtained in forward search at time ‘$t$’ and $P_r(t)$ is the shortest sub-path obtained in reverse search at time ‘$t$’. A path $p=<v_1,v_2,\ldots,v_{perm},\ldots,v_n>$ $\in V$ where ‘$v_i$’ is adjacent to ‘$v_{i+1}$’ for $1 \leq i \leq n$. ‘$v_{perm}$’ is any intermediate vertex between $v_1,\ldots,v_n$ which is visited both in forward and reverse searches. Hence, the shortest path in bidirectional search can be specified as,

$$l(P) = d_{uf} + d_{ur} + \min \sum_{i=uf}^{perm-1} l_{i,i+1}(t) + \min \sum_{i=ur}^{perm+1} l_{i,i-1}(t)$$

which minimizes the sum and minimal among all paths connecting ‘$v_1$’ to ‘$v_n$’. Here, ‘$d_{uf}$’ is the shortest path distance from source to vertex ‘$uf$’ and ‘$d_{ur}$’ is the shortest path distance from target to vertex ‘$ur$’. In addition, ‘$lf$’ is the length of the edge connecting vertex ‘$i$’ and ‘$i+1$’, and ‘$lr$’ is the length of the edge connecting vertex ‘$i$’ and ‘$i-1$’.

### 5.2.3 Modeling of Time Dependency

TDCUSPP scenarios denote the update of edge weights at some intervals of time. The method should effectively take up the changes and adjust the data structure of the network. Edge weights must be changed at regular intervals of time. This can be accomplished by the use of timers. The timer running in a separate thread ticks off and updates an edge which sets a dirty bit. The thread associated with the flag vector calculation noticing that the dirty bit is set recalculates appropriate regions/partitions and resets the dirty bit of the graph.
Figure 5.4 Code Segment for Flag Vector Calculation

Figure 5.4 displays the code for flag vector calculation. It comprises a count variable which acts as the dirty bit to determine if an edge weight has changed or not. In line 3, count = 0 indicates that no edges have been affected. Therefore, the thread waits on count as long as it is less than one. When the count exceeds one, it implies that some edge weights have changed. To determine which edge has changed, in line 4, a sync variable is used which is an index of a shared array between the Flag thread and the Timer thread. After determining which edge has changed, the region of the target vertex (variable ‘r’) is determined as in lines 5 and 6 and pre-processing is done for that particular region ‘r’ and the update time is recorded as the time taken by the pre-processing step.

The Figure 5.5 depicts the code for Timer thread. The Timer thread updates the weight of a random edge at repeated intervals of time. The edge weight changed is reflected in the sync variable as in line 10. The count variable used in line 11 denotes the number of edge-weight changes that have occurred.
The Timer thread performs this operation at repeated intervals of time. The number of edges to be changed dynamically is determined at random as in lines 7 and 8. The corresponding edges are updated as in line 10. This interval for change in edge-weight is determined by the `usleep()` function as in line 5 that makes the thread idle for an amount of time. The above code uses a 2 second timer as `usleep(2000000)`.

### 5.2.4 Simulating TDCUSPP using Parallel Programming Techniques

The two threads used for simulating TDCUSPP are also used for parallelizing the process of the shortest path computation. Here, two types of parallel programming constructs are used. They are `omp parallel` and `omp sections`.

Figure 5.6 displays the code for simulating the TDCUSPP scenario. Both Flag and Timer threads are started simultaneously and both of them run in parallel. The edges changed by Timer thread are appropriately updated by the Flag thread as explained in Figure 5.3 and Figure 5.4.
Further, these speedup techniques for TDCUSPP can be combined to improve the performance of the system. The speedup techniques considered for the shortest path computation in time-dependent dynamic environment are Arc-flags (Arc), Arc-flags and RSB (Arc-RSB) and Bidirectional search and RSB (Bi-RSB).

5.3 ARC-FLAG METHOD AND RECURSIVE SPECTRAL BISECTION PARTITIONING (Arc-RSB)

5.3.1 Overview

In this section, a new combination policy is adapted for time-dependent dynamic graphs which will take best practices of the arc-flag method and the RSB partitioning strategy. The arc-flag technique based on a grid partitioning strategy with arc-flag vectors functions effectively for all types of real world networks [9, 46]. The work proposed in [66, 71] uses a Recursive Spectral Bisection (RSB) strategy to optimize node assignment, updation of edges and to improve query performance. Here, the graph partitioning technique RSB is combined with the arc-flag method and with the bidirectional search. The performance of the same is compared with the arc-flag method.
5.3.2 Design

The arc-flag combined with RSB(Arc-RSB) consists of the pre-processing phase, the fiedler vector calculation phase and the shortest path computation phase. In the pre-processing phase, the graph is partitioned using the RSB partitioning strategy. The partitions obtained from RSB are considered to be domain specific and natural partitions which are based on the degree of vertices. The fiedler vector obtained from RSB method is used as flag vector for the arc-flag method.

The RSB Method [71] is a graph-partitioning strategy based on the degree of vertices. The partitions produced by the method are logical partitions. Every partition consists of vertices having nearly the same degree. The shortest path routing is achieved by correcting the distance label of the boundary vertices of the partitions. The graph is partitioned using RSB and later, the fiedler vector is converted into arc grids. The basic idea behind RSB is to speedup the fiedler vector computation by constructing a series of successively smaller contracted graphs that maintain the global structure of the original graph. The Fiedler vector of the smallest graph is found quickly with the Lanczos algorithm.

Now, this Fiedler vector is converted into $m \times n$ arc-flag grids. The goal of arc-flag technique in dynamic environment is to update arc labels without re-computation from scratch. Arc-Flags are set according to the fiedler vectors by considering all shortest path trees rooted at each boundary node. Hence, a possible approach is to maintain the shortest path trees for all the boundary nodes of the graph by using the dynamic algorithm. Given the huge number of boundary nodes in large graphs, this approach is impractical due to its memory overhead and time complexity. However, this method will guarantee optimal query performance (compared to a full re-computation) since it maintains exact shortest paths and changes flags only where needed. Hence, the goal is to update arc-flags without storing too much additional data. Therefore, a small efficiency loss in the query phase is accepted.
The overall procedure of combined arc-flag and RSB partitioning is given in Figure 5.7.

**STEPS:**

**RSB:**
1. The Laplacian graph is constructed.
2. Second largest Eigenvalue is found out and the corresponding Eigenvector (Fiedler vector) is found out.
3. Sorting of the Eigenvector is done.

**ARC-FLAG:**
1. Fiedler vector is converted into $m \times n$ arc-flag grids.
2. Every arc (edge) of the graph has a flag-vector associated with it. The regions are numbered from 1 to $m \times n$. Flag-vector has $m \times n$ bits.
3. Bit 1 in the flag vector for an arc $a \in A$ (edge set) indicates ‘$a$’ is on a shortest path to vertices in the corresponding region $1 \leq r \leq m \times n$. Bit 0 indicates lack of such a shortest path.

Figure 5.7 Combination of Arc-flag Method and RSB

The input to RSB procedure is the Graph $G$. The output is the sorted Fiedler vector. This will be the input to the arc-flag method. In the shortest path computation, the arc-flags will be updated and correspondingly, the shortest path will be retrieved. The computation time required for computing fiedler vector will be comparatively less than the time required for data structure updates in arc-flagged Dijkstra [28].

### 5.3.3 Analysis

The time required for constructing Laplacian matrix is $O(n)$ where ‘$n$’ is the number of vertices of the graph. To find the eigenvector corresponding to the second smallest eigenvalue, the Lanczos algorithm can be employed. This is an iterative algorithm which requires $O(n)$ operations per iteration. As the algorithm converges
in ‘m’ iterations, the algorithm has a complexity of \(O(mn)\). However, the Lanczos algorithm requires the storage of ‘m’ Lanczos vectors of length ‘n’. Ideal behavior for a serial sort of eigenvectors is \(O(n)\). The update process of arc-flags requires \(O((n+m) \cdot r)\) computational time as it performs ‘r’ times a search of graph \(G_{i-1}\).

Further, the performance and complexity of this approach are dependent on the size of the network and the number of processors used. The usage of parallel speedup technique improves the system performance by the efficient processing of the shortest path queries. Hence, the Arc-RSB technique is parallelized using OpenMP constructs to optimize the query execution time.

5.4 PARALLELIZED ARC-FLAG METHOD AND RSB PARTITIONING (Para-Arc-RSB)

5.4.1 Overview

The computational time required in the pre-processing phase by arc-flags [28, 29] shows that for computing Road-Signs along with arc-flags [47, 48], it requires about twice the computational time required for computing arc-flags, which is a small overhead compared to the speedup gained in the updating phase. This limitation can be overcome using the parallel programming constructs in the pre-processing and update phases so that the computational time can be reduced.

5.4.2 Design

The parallelized Arc-RSB consists of the pre-processing phase, the flag vector calculation and the shortest path computation. In the pre-processing phase, graph \(G\) is initialized, partitioned and flag vector is initialized in parallel using the for directive. This parallel construct will create a team of threads for individual process of initialization and partitioning and will execute each process in parallel. Here, the Dijkstra’s node selection operation is sequential where the team of threads will merge and will execute the sequential process of node selection from priority queue. In the shortest path computation
phase, fiedler vector updation and the update phases are processed in parallel using the *for* directive. The tasks identified in Para-Arc-RSB are tabulated in Table 5.1.

Table 5.1 Tasks in Para-Arc-RSB

<table>
<thead>
<tr>
<th>Sl. No.</th>
<th>Task No.</th>
<th>Task</th>
<th>Serial (S) / Parallel (P)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>T₁</td>
<td>Start or Input</td>
<td>S</td>
</tr>
<tr>
<td>2.</td>
<td>T₂</td>
<td>Graph Initialization</td>
<td>P</td>
</tr>
<tr>
<td>3.</td>
<td>T₃</td>
<td>RSB Partitioning</td>
<td>P</td>
</tr>
<tr>
<td>4.</td>
<td>T₄</td>
<td>FiedlerVector Initialization</td>
<td>P</td>
</tr>
<tr>
<td>5.</td>
<td>T₅</td>
<td>Node Selection</td>
<td>S</td>
</tr>
<tr>
<td>6.</td>
<td>T₆</td>
<td>Fiedler Vector Update</td>
<td>P</td>
</tr>
<tr>
<td>7.</td>
<td>T₇</td>
<td>Update phase</td>
<td>P</td>
</tr>
<tr>
<td>8.</td>
<td>T₈</td>
<td>Stop</td>
<td>S</td>
</tr>
</tbody>
</table>

The tasks identified from Para-Arc-RSB technique are numbered as T₁, T₂,…and T₈. The tasks which can be operated using serial programming model are marked as ‘S’. ‘P’ is used to mark the tasks executed by using parallel programming. The task graph model given in Figure 5.8 for Para-Arc-RSB depicts the various phases identified using fork and join programming model and the dependencies among the tasks. The tasks T₂, T₃, T₄, T₆ and T₇ are parallel tasks which can be executed using fork and join programming model.

These tasks will be operated/performed individually internally in parallel using the *for* directive. The serial programs/phases are (i) the start phase or the input phase, (ii) the node selection from priority queue and (iii) the stop phase which will be operated sequentially. The dependencies between the tasks are marked by arrowed lines.
5.4.3 Analysis

The time required for constructing Laplacian matrix will be shared by $\sum_{i=1}^{k} P_i$ processors which is equal to $\Theta(n/k)$, where ‘$k$’ is the number of processors. The Lanczos algorithm can be used to find the eigenvector corresponding to the second smallest eigenvalue. This is an iterative algorithm for finding eigenvector, which requires $\Theta(n/k)$ operations per iteration. However, the Lanczos algorithm requires the storage of ‘$m$’ Lanczos vectors of length ‘$n$’. Ideal behavior for a serial sort of eigenvectors is $O(n)$, and for optimal parallel sorting it is $O(\log n)$. The update operation requires $O((n + m) \cdot r)$ computational time which will be shared by $\sum_{i=1}^{k} P_i$ processors.
5.5 BIDIRECTIONAL SEARCH AND RSB PARTITIONING (Bi-RSB)

5.5.1 Overview

In the Arc-RSB pre-processing phase, additional cost is incurred for computing arc-flag vector. This additional cost can be reduced by using bidirectional search directly over the partitioned graph. Pre-processing is a process which takes longer time in this technique. Hence, a new technique Bidirectional search combined with RSB (Bi-RSB) is proposed to reduce the time of the update process and improve the query performance.

5.5.2 Design

The graph is partitioned using RSB in the pre-processing phase and in the shortest path computation phase, the bidirectional strategy is executed in each sub-graph. The two partitions are assigned in such a way that source is in the first partition and destination is in the second partition. The steps involved in solving the Bi-RSB technique are given in the Figure 5.9.

<table>
<thead>
<tr>
<th>STEPS:</th>
</tr>
</thead>
<tbody>
<tr>
<td>RSB:</td>
</tr>
<tr>
<td>1. The Laplacian graph is constructed.</td>
</tr>
<tr>
<td>2. Second largest Eigenvalue is found out and the corresponding Eigenvector (Fielder vector) is found out.</td>
</tr>
<tr>
<td>3. Sorting of the Eigenvector is done.</td>
</tr>
<tr>
<td>4. Bisecting the graph into two sub graphs $s_1$ and $s_2$ using the sorted fielder vector</td>
</tr>
<tr>
<td>BIDIRECTIONAL SEARCH:</td>
</tr>
<tr>
<td>1. The forward search is carried out in the sub-graph $s_1$ and reverse search is carried out in sub-graph $s_2$.</td>
</tr>
<tr>
<td>2. In forward search, the vertices which are reached from the source is added to the settled vertex set. Similarly, it is done in the reverse search.</td>
</tr>
<tr>
<td>3. The search is stopped when a common vertex is found in the settled vertex set.</td>
</tr>
</tbody>
</table>

Figure 5.9 Procedure for Combined Bidirectional Search and RSB Partitioning
The input to RSB procedure is the Graph G. The output is the sorted fiedler vector, which bisects the graph into two sub graphs ‘s1’ and ‘s2’. Then the search procedure will start in two sub graphs at the same time. Whenever ‘s’ is in ‘s1’ and ‘d’ is ‘s2’, this method optimizes the searching procedure. Now, sub-graph ‘s1’ will be the forward graph and ‘s2’ will be the reverse graph. Then, the graph will be searched both from source ‘s’ and destination ‘d’ using the bidirectional search. This will be the input to the arc-flag method. In the shortest path computation, the arc-flags will be updated and correspondingly, the path will be retrieved.

5.5.3 Analysis

The time required for constructing Laplacian matrix is \(O(n)\). The Lanczos algorithm determines the eigenvector corresponding to the second smallest eigenvalue. This is an iterative algorithm which requires \(O(n)\) operations per iteration. The algorithm normally converges in ‘m’ iterations. Thus, the algorithm has a complexity of \(O(mn)\). The process of sorting ‘n’ eigen vectors take a time of \(O(n)\). Searching these n vectors using bidirectional search takes two searching policies. The search tree in both directions expands with a branching factor ‘b’ and the distance from source to target is ‘l’ in traditional Dijkstra. Each of the search will be having a complexity of \(O(b^{l/2})\) and the total search time of bidirectional search will be much less than \(O(b^{l})\) in traditional Dijkstra which is equivalent to \(O(n+m \log n)\) with Fibonacci heaps. If each queue operation takes \(O(n+m \log n)\) time, the expected running time is \(O(\sqrt{n}+m \log n)\).

5.6 PARALLELIZED BIDIRECTIONAL SEARCH AND RSB PARTITIONING (Para-Bi-RSB)

5.6.1 Overview

The Bi-RSB technique reduces the pre-processing time and updation time. This reduction in processing time can be improved by parallelizing the individual processes of pre-processing and update phases using the OpenMP constructs. This Para-Bi-RSB consists of mainly two processing phases. They are parallelized pre-processing phase where RSB partitioning will be carried out in parallel and parallelized shortest path
computation phase where the bidirectional search will be processed in parallel using the `omp parallel` and `omp sections`.

### 5.6.2 Design

The Lanczos algorithm for fiedler vector calculation and the bidirectional part of solving the shortest path queries are parallelized. The speedup for QPL is similar in behaviour with respect to time-dependent arc-flags. The parallelized Bi-RSB consists of the pre-processing phase, the flag vector calculation and the shortest path computation. In the pre-processing phase, the graphs FG and RG are initialized and partitioned in parallel using the `for` directive. This parallel construct will create a team of threads and will execute each process in parallel bidirectionally using the `sections` directive. Here, the Dijkstra’s node selection operation is sequential where the team of threads will merge and will execute the sequential process of node selection from priority queue. In the shortest path computation phase, the update phase is processed in parallel using the `for` directive and the `sections` directive both for FG and RG. The tasks identified in Para-Bi-RSB are tabulated in Table 5.2.

#### Table 5.2 Tasks in Para-Bi-RSB

<table>
<thead>
<tr>
<th>Sl. No.</th>
<th>Task No.</th>
<th>Task</th>
<th>Serial (S) / Parallel (P)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>T₁</td>
<td>Start or Input</td>
<td>S</td>
</tr>
<tr>
<td>2.</td>
<td>T₂</td>
<td>RSB Partitioning</td>
<td>P</td>
</tr>
<tr>
<td>3.</td>
<td>T₃</td>
<td>FG – Initialization</td>
<td>P</td>
</tr>
<tr>
<td>4.</td>
<td>T₄</td>
<td>RG – Initialization</td>
<td>P</td>
</tr>
<tr>
<td>5.</td>
<td>T₅</td>
<td>FG – Node Selection</td>
<td>S</td>
</tr>
<tr>
<td>6.</td>
<td>T₆</td>
<td>RG – Node Selection</td>
<td>S</td>
</tr>
<tr>
<td>7.</td>
<td>T₇</td>
<td>FG – Update phase</td>
<td>P</td>
</tr>
<tr>
<td>8.</td>
<td>T₈</td>
<td>RG – Update phase</td>
<td>P</td>
</tr>
<tr>
<td>9.</td>
<td>T₉</td>
<td>Stop</td>
<td>S</td>
</tr>
</tbody>
</table>

The task graph model given in Figure 5.10 for Para-Bi-RSB depicts the various phases identified using fork and join programming model and the dependencies among these tasks. The phases in Para-Bi-RSB that can be operated in parallel are RSB partitioning, graph initialization (FG and RG) and path updation phase. These phases will
be performed individually in parallel using the *for* directive. In addition, the operations relevant for FG and RG are carried in parallel using the *sections* directive. The serial programs/phases are (i) the start phase or the input phase, (ii) the node selection from priority queues Q and rQ and (iii) the stop phase. The dependencies between the tasks are marked by arrowed lines. The task graph representation in this work is carried out in parallel both internally (*for* directive) and externally (*sections* directive).

RSB partitioning is typically used as a pre-processing step prior to running the shortest computation phase on a multi-core computer. They usually run on multiple processors resting in the same system and the results of multiple processors are communicated to the shared memory computer. As it is the fast method to re-partition the graph in shared memory systems, the Para-Bi-RSB technique is the recommended technique for real-time applications.

![Task Graph of Para-Bi-RSB](image)

Figure 5.10 Task Graph of Para-Bi-RSB
5.6.3 Analysis

Depending on the number of processors running in the system, the iterative operations will be parallelly executed using the multithreaded approach. Each process \( P_i \) \((1 \leq i \leq k)\) (‘k’ be the number of processors in the system) initializes the forward priority queue \( Q \) and reverse priority queue \( rQ \) partly. For large value of ‘\( n \)’, the operation of initialization phase and update phase are shared by multiple processors which take a time ‘\( t \)’ and is less than the time achieved in sequential bidirectional search. If the node size is very less, then the time for communication between threads is consumed by the computation time. Here too, in the worst case, the running time will be less than the time achieved in sequential bidirectional search i.e. \( O(\sqrt{n} + m \log n) \).

5.7 **EMPIRICAL STUDY OF SHORTEST PATH QUERIES IN DYNAMIC NETWORKS**

Empirical study includes the phases of study definition, study planning and operation and interpretation. They are discussed in the following sub-sections.

5.7.1 Study Definition

The motivation of this study is formulated with the following key questions:

- How far the output parameters and metrics obtained are comparable with the values obtained in other domains?
- How far do the nodes and edges generated play a vital role in the output metrics?
- How far the parallel speedup techniques are helpful? Will these techniques improve the system performance?
- Which speedup technique will be suitable for each type of domain specific dynamic networks?

5.7.2 Study Planning and Operation

The speedup techniques are implemented and tested in random, planar graphs and real world graphs. The real world graphs are extracted from Tamil Nadu map OSM file.
The measured output parameters are the pre-processing time and the update time. The estimated metric is QPL which determines the usability of the speed-up technique in the time-dependent scenario. For all categories of input graph types, the nodes ranging from 1000 to 10000 are considered. The pre-processing time and the update time are calculated and then the output metric query performance loss is evaluated.

**System Configuration**

Computation of the shortest path in dynamic networks using Arc-RSB, Bi-RSB, Para-Arc-RSB and Para-Bi-RSB are implemented and tested in the following system configuration. All results are compiled on an Intel Core 2 Duo Desktop with clock speed 2.83 GHz with 4GB RAM running on Ubuntu 10.04 Linux. The programs are written in C++ and Library of Efficient Data types and Algorithms (LEDA) [91]. LEDA is meant for effectively using the graph data structures and library files relevant to Dijkstra’s algorithm.

**Study Limitations**

This study consists of graphs of limited size (i.e. maximum node size=10000). The system configuration is also given above. The developed speedup techniques are tested in the same system for the entire work. The generated edges for each types of graphs are tabulated in Table 5.3.

<table>
<thead>
<tr>
<th>Table 5.3 Nodes and Edges Generated for All Types of Graphs</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Random Graphs</strong></td>
</tr>
<tr>
<td>Nodes</td>
</tr>
<tr>
<td>Edges</td>
</tr>
<tr>
<td><strong>Planar Graphs</strong></td>
</tr>
<tr>
<td>Nodes</td>
</tr>
<tr>
<td>Edges</td>
</tr>
<tr>
<td><strong>Road Networks</strong></td>
</tr>
<tr>
<td>Nodes</td>
</tr>
<tr>
<td>Edges</td>
</tr>
</tbody>
</table>
5.7.3 Results and Discussion

The results of the parameters computed are discussed in this section. The optimality of the results is statistically analyzed using ANOVA (for comparing three techniques) and T-test (for comparing two tests).

**Hypotheses with respect to the result of the parameter P: (Pre-processing time)**

Null hypothesis $H_0 : P_1 = P_2 = P_3$, where $P_1 =$ Pre-processing time in Arc-flag method in random graphs, $P_2 =$ Pre-processing time in Arc-RSB method in random graphs and $P_3 =$ Pre-processing time in Bi-RSB method in random graphs.

(There is no significant difference between the three techniques in terms of the pre-processing time obtained)

Alternative hypothesis $H_1$: Pre-processing time mean values are not equal for at least one pair of the result mean values of the parameter P.

(There is a significant difference between the three techniques in terms of the pre-processing time obtained)

**Hypotheses with respect to the result of the parameter P: (Pre-processing time)**

Null hypothesis $H_0 : P_1 = P_2 = P_3$, where $P_1 =$ Pre-processing time in Arc-flag method in planar graphs, $P_2 =$ Pre-processing time in Arc-RSB method in planar graphs and $P_3 =$ Pre-processing time in Bi-RSB method in planar graphs.

(There is no significant difference between the three techniques in terms of the pre-processing time obtained)

Alternative hypothesis $H_2$: Pre-processing time mean values are not equal for at least one pair of the result mean values of the parameter P.

(There is a significant difference between the three techniques in terms of the pre-processing time obtained)

**Hypotheses with respect to the result of the parameter P: (Pre-processing time)**

Null hypothesis $H_0 : P_1 = P_2 = P_3$, where $P_1 =$ Pre-processing time in Arc-flag method in road networks, $P_2 =$ Pre-processing time in Arc-RSB method in road networks and $P_3 =$ Pre-processing time in Bi-RSB method in road networks.
(There is no significant difference between the three techniques in terms of the pre-processing time obtained)

**Alternative hypothesis H3:** Pre-processing time mean values are not equal for at least one pair of the result mean values of the parameter P.

(There is a significant difference between the three techniques in terms of the pre-processing time obtained)

**Hypotheses with respect to the result of the parameter U: (Update time)**

**Null hypothesis H0 :** \( U_1 = U_2 = U_3 \), where \( U_1 = \) Update time in Arc method in random graphs, \( U_2 = \) Update time in Arc-RSB method in random graphs and \( U_3 = \) Update time in Bi-RSB method in random graphs.

(There is no significant difference between the three techniques in terms of the update time obtained)

**Alternative hypothesis H4:** Update time mean values are not equal for at least one pair of the result mean values of the parameter U.

(There is a significant difference between the three techniques in terms of the update time obtained)

**Hypotheses with respect to the result of the parameter U: (Update time)**

**Null hypothesis H0 :** \( U_1 = U_2 = U_3 \), where \( U_1 = \) Update time in Arc method in planar graphs, \( U_2 = \) Update time in Arc-RSB method in planar graphs and \( U_3 = \) Update time in Bi-RSB method in planar graphs.

(There is no significant difference between the three techniques in terms of the update time obtained)

**Alternative hypothesis H5:** Update time mean values are not equal for at least one pair of the result mean values of the parameter U.

(There is a significant difference between the three techniques in terms of the update time obtained)
Hypotheses with respect to the result of the parameter $U$: (Update time)

**Null hypothesis** $H_0 : U_1 = U_2 = U_3$, where $U_1 =$ Update time in Arc method in road networks, $U_2 =$ Update time in Arc-RSB method in road networks and $U_3 =$ Update time in Bi-RSB method in road networks.

(There is no significant difference between the three techniques in terms of the update time obtained)

**Alternative hypothesis** $H_6$: Update time mean values are not equal for at least one pair of the result mean values of the parameter $U$.

(There is a significant difference between the three techniques in terms of the update time obtained)

Table 5.4 ANOVA for Experimentation of Techniques I, II and III

<table>
<thead>
<tr>
<th>Graph Type</th>
<th>Technique</th>
<th>Pre-processing time</th>
<th>Update time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>I</td>
<td>II</td>
<td>III</td>
</tr>
<tr>
<td>Random</td>
<td>Arc</td>
<td>Arc-RSB</td>
<td>Bi-RSB</td>
</tr>
<tr>
<td>Planar</td>
<td>Arc</td>
<td>Arc-RSB</td>
<td>Bi-RSB</td>
</tr>
<tr>
<td>Road Network</td>
<td>Arc</td>
<td>Arc-RSB</td>
<td>Bi-RSB</td>
</tr>
</tbody>
</table>

From Table 5.4, it is concluded that the calculated significance level of the parameter pre-processing time of comparing three techniques namely, Arc, Arc-RSB and Bi-RSB always satisfy the condition ($p$ value $<$ 0.05) in random, planar and road networks. There is a significant difference between the results for different pre-processing times of the above three techniques, namely, Arc, Arc-RSB and Bi-RSB. Hence, the null hypotheses for $H_1$, $H_2$ and $H_3$ are rejected.

It is concluded that the calculated significance level of the parameter update time of comparing two techniques namely, Arc and Arc-RSB or Arc and Bi-RSB always satisfy the condition ($p$ value $<$ 0.05) in random, planar and road networks. There is a
significant difference between the results for the different update times of the above three techniques. Hence, the null hypotheses for H4, H5 and H6 are rejected.

T-test

From the previous results, it is concluded that there is a significant difference in the pre-processing time and the update time of the speedup techniques. Now it is also necessary to find the optimality of result for the speedup techniques to the shortest path problem which can be verified in terms of the pre-processing time and the update time obtained from several experiments. The existing Arc-flag technique is compared with the proposed techniques namely, Arc-RSB and Bi-RSB individually using T-test. For the various experiments, a statistical analysis is carried out in this section. The hypotheses sets are:

Hypotheses with respect to the result of the parameter P: (Pre-processing time)

Null hypothesis $H_0 : P_1 = P_2$ , where $P_1 = \text{Pre-processing time in Arc method in random graphs}$ and $P_2 = \text{Pre-processing time in Arc-RSB method in random graphs}$.
(There is no significant difference between the two techniques in terms of the pre-processing time obtained)

Alternative hypothesis $H_1$: Pre-processing time mean values are not equal for at least one pair of the result mean values of the parameter P.
(There is a significant difference between the two techniques in terms of the pre-processing time obtained)

Hypotheses with respect to the result of the parameter P: (Pre-processing time)

Null hypothesis $H_0 : P_1 = P_2$ , where $P_1 = \text{Pre-processing time in Arc method in random graphs}$ and $P_2 = \text{Pre-processing time in Bi-RSB method in random graphs}$.
(There is no significant difference between the two techniques in terms of the pre-processing time obtained)

Alternative hypothesis $H_2$: Pre-processing time mean values are not equal for at least one pair of the result mean values of the parameter P.
(There is a significant difference between the two techniques in terms of the pre-processing time obtained)

**Hypotheses with respect to the result of the parameter P: (Pre-processing time)**

Null hypothesis $H_0 : P_1 = P_2$, where $P_1 =$ Pre-processing time in Arc method in planar graphs and $P_2 =$ Pre-processing time in Arc-RSB method in planar graphs.

(There is no significant difference between the two techniques in terms of the pre-processing time obtained)

Alternative hypothesis $H_3$: Pre-processing time mean values are not equal for at least one pair of the result mean values of the parameter P.

(There is a significant difference between the two techniques in terms of the pre-processing time obtained)

**Hypotheses with respect to the result of the parameter P: (Pre-processing time)**

Null hypothesis $H_0 : P_1 = P_2$, where $P_1 =$ Pre-processing time in Arc method in planar graphs and $P_2 =$ Pre-processing time in Bi-RSB method in planar graphs.

(There is no significant difference between the two techniques in terms of the pre-processing time obtained)

Alternative hypothesis $H_4$: Pre-processing time mean values are not equal for at least one pair of the result mean values of the parameter P.

(There is a significant difference between the two techniques in terms of the pre-processing time obtained)

**Hypotheses with respect to the result of the parameter P: (Pre-processing time)**

Null hypothesis $H_0 : P_1 = P_2$, where $P_1 =$ Pre-processing time in Arc method in road networks and $P_2 =$ Pre-processing time in Arc-RSB method in road networks.

(There is no significant difference between the two techniques in terms of the pre-processing time obtained)

Alternative hypothesis $H_5$: Pre-processing time mean values are not equal for at least one pair of the result mean values of the parameter P.
There is a significant difference between the two techniques in terms of the preprocessing time obtained.

Hypotheses with respect to the result of the parameter P: (Pre-processing time)
Null hypothesis $H_0 : P_1 = P_2$, where $P_1 =$ Pre-processing time in Arc method in road networks and $P_2 =$ Pre-processing time in Bi-RSB method in road networks.
(There is no significant difference between the two techniques in terms of the preprocessing time obtained)
Alternative hypothesis $H_6$: Pre-processing time mean values are not equal for at least one pair of the result mean values of the parameter P.
(There is a significant difference between the two techniques in terms of the preprocessing time obtained)

Hypotheses with respect to the result of the parameter U: (Update time)
Null hypothesis $H_0 : U_1 = U_2$, where $U_1 =$ Update time in Arc method in random graphs and $U_2 =$ Update time in Arc-RSB method in random graphs.
(There is no significant difference between the two techniques in terms of the update time obtained)
Alternative hypothesis $H_7$: Update time mean values are not equal for at least one pair of the result mean values of the parameter U.
(There is a significant difference between the two techniques in terms of the update time obtained)

Hypotheses with respect to the result of the parameter U: (Update time)
Null hypothesis $H_0 : U_1 = U_2$, where $U_1 =$ Update time in Arc method in random graphs and $U_2 =$ Update time in Bi-RSB method in random graphs.
(There is no significant difference between the two techniques in terms of the update time obtained)
Alternative hypothesis $H_8$: Update time mean values are not equal for at least one pair of the result mean values of the parameter U.
(There is a significant difference between the two techniques in terms of the update time obtained)

**Hypotheses with respect to the result of the parameter U: (Update time)**

Null hypothesis $H_0 : U_1 = U_2$, where $U_1 =$ Update time in Arc method in planar graphs and $U_2 =$ Update time in Arc-RSB method in planar graphs.

(There is no significant difference between the two techniques in terms of the update time obtained)

Alternative hypothesis $H_9$: Update time mean values are not equal for at least one pair of the result mean values of the parameter U.

(There is a significant difference between the two techniques in terms of the update time obtained)

**Hypotheses with respect to the result of the parameter U: (Update time)**

Null hypothesis $H_0 : U_1 = U_2$, where $U_1 =$ Update time in Arc method in planar graphs and $U_2 =$ Update time in Bi-RSB method in planar graphs.

(There is no significant difference between the two techniques in terms of the update time obtained)

Alternative hypothesis $H_{10}$: Update time mean values are not equal for at least one pair of the result mean values of the parameter U.

(There is a significant difference between the two techniques in terms of the update time obtained)

**Hypotheses with respect to the result of the parameter U: (Update time)**

Null hypothesis $H_0 : U_1 = U_2$, where $U_1 =$ Update time in Arc method in road networks and $U_2 =$ Update time in Arc-RSB method in road networks.

(There is no significant difference between the two techniques in terms of the update time obtained)

Alternative hypothesis $H_{11}$: Update time mean values are not equal for at least one pair of the result mean values of the parameter U.
(There is a significant difference between the two techniques in terms of the update time obtained)

**Hypotheses with respect to the result of the parameter U: (Update time)**

Null hypothesis $H_0 : U_1 = U_2$, where $U_1 = \text{Update time in Arc method in road networks}$ and $U_2 = \text{Update time in Bi-RSB method in road networks}$.

(There is no significant difference between the two techniques in terms of the update time obtained)

Alternative hypothesis $H_{12}$: Update time mean values are not equal for at least one pair of the result mean values of the parameter U.

(There is a significant difference between the two techniques in terms of the update time obtained)

As per this experimental design, the T-test results are shown in Table 5.5, the inferences are as listed below at a significant level $=0.05$.

**Table 5.5 T-test for Experimentation**

<table>
<thead>
<tr>
<th>Graph Type</th>
<th>Technique</th>
<th>Pre-processing time</th>
<th>Update time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Hypothesis</td>
<td>p value</td>
</tr>
<tr>
<td>Random</td>
<td>Arc</td>
<td>I</td>
<td>H1</td>
</tr>
<tr>
<td></td>
<td>Arc-RSB</td>
<td>II</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Arc</td>
<td>I</td>
<td>H2</td>
</tr>
<tr>
<td></td>
<td>Bi-RSB</td>
<td>II</td>
<td></td>
</tr>
<tr>
<td>Planar</td>
<td>Arc</td>
<td>I</td>
<td>H3</td>
</tr>
<tr>
<td></td>
<td>Arc-RSB</td>
<td>II</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Arc</td>
<td>I</td>
<td>H4</td>
</tr>
<tr>
<td></td>
<td>Bi-RSB</td>
<td>II</td>
<td></td>
</tr>
<tr>
<td>Road Network</td>
<td>Arc</td>
<td>I</td>
<td>H5</td>
</tr>
<tr>
<td></td>
<td>Arc-RSB</td>
<td>II</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Arc</td>
<td>I</td>
<td>H6</td>
</tr>
<tr>
<td></td>
<td>Bi-RSB</td>
<td>II</td>
<td></td>
</tr>
</tbody>
</table>

From Table 5.5, it is concluded that the calculated significance level of the parameter pre-processing time of comparing Arc and Arc-RSB or Arc and Bi-RSB techniques, it always satisfies the condition (p value $< 0.05$) except in few cases. There is a significant difference between the results for the different pre-processing times of two techniques, namely, Arc method and Arc-RSB method or Arc and Bi-RSB. Hence, the
null hypotheses for H1, H2, H3, H4 and H6 are rejected. But, the same two techniques in road networks give a significance level which does not satisfy the condition (p-value < 0.05). Hence, the null hypothesis for H5 is accepted.

It is concluded that the calculated significance level of the parameter update time of comparing the techniques Arc and Arc-RSB or Arc and Bi-RSB always satisfy the condition (p value < 0.05) except in a few cases. There is a significant difference between the results for the different update times of two techniques, namely, Arc method and Arc-RSB method or Arc and Bi-RSB. Hence, the null hypotheses for H8, H9, H10, H11 and H12 are rejected. But the same two techniques in random networks gives a significance level which does not satisfy the condition (p-value<0.05). Hence, the null hypothesis for H7 is accepted.

Hence, except in the pre-processing time of Arc and Bi-RSB in road networks and update of Arc and Arc-RSB, other techniques show a significant difference in their processing times. Further, it is also required to determine the speedup technique which has the minimum pre-processing time and update time. This is analyzed using descriptive statistics given in Table 5.6 and Table 5.7. This decrease in processing time will stimulate efficient query execution. Table 5.6 shows the descriptive statistics (the maximum, minimum, median, mean values and standard deviation) of each of the speedup technique for the parameter pre-processing time.

Table 5.6 Descriptive Statistics of Pre-processing Time Measures

<table>
<thead>
<tr>
<th>Graph Type</th>
<th>Random Graph</th>
<th>Planar Graph</th>
<th>Road Networks</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Technique</strong></td>
<td><strong>Max, Min</strong></td>
<td><strong>Mean, Median</strong></td>
<td><strong>Standard Deviation</strong></td>
</tr>
<tr>
<td>Arc</td>
<td>71.13 0.210</td>
<td>17.99 6.96</td>
<td>24.69 81.5</td>
</tr>
<tr>
<td>Arc-RSB</td>
<td>233.1 1.62</td>
<td>84.5 59.9</td>
<td>81.5 0.805</td>
</tr>
<tr>
<td>Bi-RSB</td>
<td>2.330 0.0300</td>
<td>0.907 0.720</td>
<td>0.805 0.085</td>
</tr>
<tr>
<td>Arc</td>
<td>24.52 0.240</td>
<td>9.36 7.38</td>
<td>8.38 51.3</td>
</tr>
<tr>
<td>Arc-RSB</td>
<td>147.2 1.10</td>
<td>56.0 43.0</td>
<td>51.3 0.808</td>
</tr>
<tr>
<td>Bi-RSB</td>
<td>2.360 0.0200</td>
<td>0.905 0.710</td>
<td>0.808 0.0808</td>
</tr>
<tr>
<td>Arc</td>
<td>47.69 0.250</td>
<td>16.47 10.64</td>
<td>16.79 7.97</td>
</tr>
<tr>
<td>Arc-RSB</td>
<td>24.71 0.350</td>
<td>9.22 8.18</td>
<td>7.97 0.710</td>
</tr>
<tr>
<td>Bi-RSB</td>
<td>2.350 0.0300</td>
<td>0.905 0.710</td>
<td>0.710 0.0803</td>
</tr>
</tbody>
</table>
From Table 5.6, it is evident that there is an increase in mean pre-processing time in Arc-RSB. The same mean pre-processing time is reduced in Bi-RSB from which it can be concluded that reduced pre-processing time in Bi-RSB will improve the query performance. In other words, any technique that is independent of edge weights in its pre-processing phase is more suitable for dynamic networks and hence, the RSB partitioning is better than the grid partitioning.

Table 5.7 shows the descriptive statistics (the maximum, minimum, median, mean values and standard deviation) of each of the speedup techniques for the parameter update time.

<table>
<thead>
<tr>
<th>Graph Type</th>
<th>Random Graph</th>
<th>Planar Graph</th>
<th>Road Networks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Technique</td>
<td>Max, Min</td>
<td>Mean, Median</td>
<td>Standard Deviation</td>
</tr>
<tr>
<td>Arc</td>
<td>57.19</td>
<td>13.97</td>
<td>19.56</td>
</tr>
<tr>
<td>Arc-RSB</td>
<td>8.026, 0.164</td>
<td>2.958, 5.25</td>
<td>2.824</td>
</tr>
<tr>
<td>Bi-RSB</td>
<td>0.0, 0.0</td>
<td>0.0, 0.0</td>
<td>0.0, 0.0</td>
</tr>
<tr>
<td>Arc</td>
<td>18.30, 0.176</td>
<td>7.01, 5.50</td>
<td>6.28</td>
</tr>
<tr>
<td>Arc-RSB</td>
<td>5.418, 0.039</td>
<td>1.656, 1.534</td>
<td>1.625</td>
</tr>
<tr>
<td>Bi-RSB</td>
<td>0.0, 0.0</td>
<td>0.0, 0.0</td>
<td>0.0, 0.0</td>
</tr>
<tr>
<td>Arc</td>
<td>0.006900</td>
<td>0.002650</td>
<td>0.002104</td>
</tr>
<tr>
<td>Arc-RSB</td>
<td>0.8970</td>
<td>0.3431</td>
<td>0.2960</td>
</tr>
<tr>
<td>Bi-RSB</td>
<td>0.0140</td>
<td>0.2635</td>
<td>0.0</td>
</tr>
</tbody>
</table>

The update time in Arc-RSB is reduced in all types of graphs except road networks. The update time in Bi-RSB is nullified due to the non-availability of data structures. It can be concluded that the efficiency loss in the query execution will be minimized.

Table 5.8 shows the descriptive statistics (the median, mean values and standard deviation) of each of the speedup techniques for the metric QPL.
Table 5.8 Descriptive Statistics of QPL Metric

<table>
<thead>
<tr>
<th>Graph Type</th>
<th>Random Graph</th>
<th>Planar Graph</th>
<th>Road Networks</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Arc</td>
<td>Arc-RSB</td>
<td>Bi-RSB</td>
</tr>
<tr>
<td>Mean, Median</td>
<td>0.75</td>
<td>0.035</td>
<td>0.005</td>
</tr>
<tr>
<td>Standard Deviation</td>
<td>0.009</td>
<td>0.004</td>
<td>0.005</td>
</tr>
</tbody>
</table>

It is evident from Table 5.8, that the Bi-RSB gives a reduced QPL in all types of graphs compared to the Arc method and the Arc-RSB method when the mean values of these techniques are compared except road networks. In the Arc-RSB method, the size of flag vector is reduced as the numbers of regions are reduced. Therefore, the mean QPL is reduced in random and planar graphs. In Bi-RSB, the nullified update time decides the QPL value. In dynamic graphs, the RSB based speedup techniques achieve a better performance than grid partitioning of the Arc method by reducing the update time and QPL.

**Parallelized Speedup Techniques**

The parallelized speedup techniques Para, Para-Arc-RSB and Para-Bi-RSB are compared with its own sequential counterparts for QPL metric. The comparison shows that the speedup due to parallelism is achieved in all the techniques in all the graph types.
In random graphs, the QPL values are reduced by 29% in Para-Arc method, 71% in Para-Arc-RSB method and 31% in Para-Bi-RSB method than its sequential counterparts.

Figure 5.11 Comparison of Parallelized Speedup Techniques in Random Graphs
The parallelized speedup techniques in planar graphs show reduced QPL values of 27% in arc-flag method, 60% in Arc-RSB method and 38% in Bi-RSB method.

When parallelizing the speedup techniques, the QPL results are better monitored in road networks. The QPL values are reduced by 68% in the Arc method, 60% in the Arc-RSB method and 82% in the Bi-RSB method.
5.8 SUMMARY

In this part of research work, two new speedup techniques for dynamic networks namely, Arc-RSB and Bi-RSB are presented for random, planar and road networks. In random and planar graphs, Bi-RSB outperforms the other speedup techniques by reduced pre-processing time, update time and QPL. It also highlights that the amount of pre-processing becomes the actual overhead when solving dynamic networks.

In addition, these techniques are parallelized and new parallelized speedup techniques namely, Para-arc-RSB and Para-Bi-RSB for multi-core systems are also presented. These speedup techniques are capable of processing shortest path queries in time dependent dynamic networks. The major contribution of RSB partitioning technique is the nullified update time which is an inherent parameter related to the efficient query performance. All parallelized speedup techniques show improved performance when compared to the sequential speedup techniques.