Journal of King Saud University – Computer and Information Sciences (2014) 26, 111–119

King Saud University
Journal of King Saud University – Computer and Information Sciences
www.ksu.edu.sa
www.sciencedirect.com

ORIGINAL ARTICLE

A speedup technique for dynamic graphs using partitioning strategy and multithreaded approach

R. Kalpana [a,*], P. Thambidurai [b]

[a] *Dept. of CSE, Pondicherry Engineering College, Puducherry, India*
[b] *Dept. of CSE, Perunthalaivar Kamarajar Institute of Engg. & Tech, Karaikal, Puducherry, India*

Received 30 October 2012; revised 18 May 2013; accepted 22 June 2013
Available online 1 July 2013

KEYWORDS
Dijkstra's algorithm;
Speedup techniques;
Dynamic graphs;
Parallel programming

Abstract There are many pre-processing-based speedup techniques for shortest path problems that are available in the literature. These techniques have an increased demand because of large datasets in such applications such as roadmaps, web search engines and mobile data sets. Pre-processing for the Time-Dependent Shortest Path Problem is still a demanding process that involves graph or network partitioning strategy. Efficient pre-processing of graphs or networks reduces the shortest path computation time while parallelizing the pre-processing phase improves the speedup of the system. In this paper, a speedup technique called Recursive Spectral Bisection (RSB) combined with the Elliptic Convolution of the shortest path method is proposed for dynamic Time-Dependent networks. The same method has been parallelized, and the results are tested on three types of graphs. It is observed that the Time-Dependent RSB combined with the Elliptic Convolution of the shortest path method has no update time, and the Query Performance Loss (QPL) is reduced in planar and road networks compared to random networks. In road networks, the proposed method achieves an average speedup in a QPL of 140. The use of the Parallel speedup technique results in an average speedup in a QPL of more than 1 in the planar and road networks.

1. Introduction

The computation of shortest paths from a given source to a specific destination has the largest scope in the present scenario of real-world applications. The most famous applications are route planning systems for cars, bikes and hikers and timetable information systems for scheduled vehicles, such as trains and buses. If such a system is realized as a central server, it must answer a large number of customer queries in which the customers ask for their best itineraries. Users of such a system continuously enter requests for finding their "best" connections. In addition, similar queries appear as sub-problems in line planning, timetable generation, tour planning, logistics, and traffic simulations. The algorithmic core problem that underlies the above scenario is a special case of the single-source shortest path problem on a given directed graph with non-negative edge lengths. The shortest path queries for such applications were originally solved by Dijkstra, Bellman-Ford, and Johnson. Dijkstra's algorithm implemented with Fibonacci heaps is still the fastest known algorithm for the general

[*] Corresponding author. Tel.: +91 413 2655282.
E-mail address: rkalpana@pec.edu (R. Kalpana).

1319-1578 © 2013 Production and hosting by Elsevier B.V. on behalf of King Saud University.
http://dx.doi.org/10.1016/j.jksuci.2013.06.002

case of arbitrary nonnegative edge lengths, taking $O(m + n \log n)$ worst-case time. In real-world applications, a layout of the graph is given as an input, and the specific graphs to be considered are notably large. Moreover, the number of queries to be processed within a short time is also notably large. This problem motivates the use of speedup techniques for shortest-path computations (Wagner and Willhalm, 2007).

The main focus of the speedup techniques is to reduce the runtime of the on-line queries (Wagner and Willhalm, 2007). The speedup techniques for the Dijkstra's algorithm can reduce the running time and often result in a sub-linear running time. They crucially depend on the fact that the Dijkstra's algorithm is label setting and that it can be terminated when the destination node is determined. Therefore, the algorithm does not necessarily search the whole graph. Finding the shortest path in a dynamic network is the hottest topic of interest for real-time applications such as vehicle routing, route planning, web searching, and applications in which the edge weight of the graph changes randomly based on the parameters that are being considered. In dynamic road networks, there is always a need for such speedup techniques. In addition, dynamic road networks are of interest from an industrial point of view, and they are usually more accurate models than static networks. Many of these speedup techniques are built on Dijkstra's algorithm and heuristically improve their performance while maintaining their correctness, both in static and dynamic environments.

The focus is now moving to provide such speedup techniques for the Time-Dependent Shortest Path Problem (TDSPP), which constitutes the SPP applied on a Time-Dependent network. The Time-Dependent Shortest Path Problem (TDSPP) is a dynamic graph problem that is NP-hard and non-linear (Li et al., 2005).

In this paper, a new speedup technique is proposed that uses a new partitioning strategy. In addition, this new speedup technique will eliminate the update time and improve the Query Performance Loss (QPL). A Recursive Spectral Bisection partitioning strategy is incorporated in a pre-processing phase, and the Elliptic Convolution of the shortest path method is incorporated in the shortest path computation phase. This new technique is parallelized using parallel programming constructs, and a new speedup technique is proposed. The results of these new techniques are tested in random, planar and road networks.

The remainder of this paper is organized as follows. Section 2 briefly explains the studies that are related to dynamic graph speedup techniques. Section 3 describes the Time-Dependent Shortest Path Problem and the techniques that are used for the shortest path computation. Section 4 provides a detailed experimental evaluation of the proposed new techniques and analyzes the results. Section 5 discusses the concluding remarks.

2. Related work

The arc-flag method is the most popular speedup technique in dynamic environments (Berrettini et al., 2009; D'Angelo et al., 2011, 2012). There are a number of improvements of the basic variant of the arc-flag acceleration for point-to-point shortest path computations on large graphs (Berrettini et al., 2009; D'Angelo et al., 2011). In the case of a dynamic scenario, the changes in arc weight and also the recomputation of the pre-processing phase will be conducted.

There are voluminous contributions on Time-dependent networks. Both FIFO and Non-FIFO Time-dependent shortest problems (Ding et al., 2008) consider both directed and undirected graphs. The computation effort is reduced using an A^* search in bidirectional core-based routing (Delling and Nannicini, 2008) in Time-dependent dynamic environments. The idea behind bidirectional core-based routing (Delling and Nannicini, 2008) is to shrink the original graph to obtain a new core graph with a smaller number of vertices. This action reduces the search process because most of the time the search is conducted on the core. Landmark-based routing (Delling and Wagner, 2007) uses the minimum weight of each edge to compute the distance labels, which calculates the estimated departure time by altering the priority of a node. In Core-ALT (Delling and Nannicini, 2008) routing, an initial contraction step prior to ALT pre-processing must be performed. Landmarks are then chosen from the core and stored with a distance value. In geometric containers (Wagner and Willhalm, 2003), the search space of the algorithm can be reduced by extracting geometric information from a given layout of the graph and by encapsulating precomputed shortest-path information in the geometric containers. In a Goal-directed search, also called A^* (Delling and Nannicini, 2008), the algorithm pushes the search toward a target by adding a potential to the priority of each node. The usage of Euclidean potentials requires no pre-processing. For the use of an arc-flag approach in a dynamic scenario, a new algorithm for update operations that is subject to edge weight decrease operations (D'Angelo et al., 2012) decreases the pre-processing time and the low consumption of the space. A new data structure Road-sign (D'Angelo et al., 2011), which is used in the pre-processing phase of the arc-flag approach, helps to update the edges effectively and reduces the space consumption. The benefits of dynamic time-dependent planning (Ehmke and Mattfeld, 2010), in contrast to common static planning methods, are to achieve better quality results in real-time applications. Traffic data of City logistics show data allocation models that can be applied to any real data sets. The recursive spectral bisection (Zhang et al., 2010) technique is used as an iterative labeling algorithm for network decomposition because this approach is usually used to solve the problem in transportation applications. Here, the network partitioning is made based on the domain, and thus, there is no need to update the data structures.

Currently, every real-world application runs on a multi-core processor that has multiple cores that communicate via shared memory. The multi-core processor model (Sibai, 2013) presents the architecture of core processors and how data partitions are made on core processors. The efficiency of the applications that run on multi-core processors can be improved when **the** parallel Gaussian elimination method is used.

3. Time-Dependent Shortest Path Problem

This section addresses the speedup techniques for shortest path computations in a dynamic environment. The partitioning techniques' recursive spectral bisection method is considered for better improvement in a time-dependent dynamic environment. This new RSB-based partitioning technique dominates the existing arc-flag-based partitioning technique (Mohring et al., 2006) in its output parameter update time. The input graph representation is of different types: random graphs, planar graphs, and road networks (which are considered to be dynamic in nature). Two types of threads are used to make the graphs dynamic. These threads are the Flag Thread and the Timer Thread. The Flag Thread checks whether the edge weight has been changed or not. The Timer thread is used to change the edge weight at a random time, to make the graph dynamic.

This section addresses the modeling of TDSPP, TD-Combined RSB and Elliptic Convolution of the shortest path method, Parallel TD-Combined RSB and Elliptic Convolution of the shortest path method.

3.1. Modeling of Time Dependency

Time-dependent scenarios involve the update of edge weights at certain intervals of time. This method should effectively address the changes and adjust the data structure. Edge weights must be changed at regular intervals of time. This change can be accomplished by the use of timers. The timer runs in a separate thread and ticks off and updates an edge (sets a dirty bit). The thread associated with the Flag vector calculation notices that the dirty bit is set and recalculates appropriate regions and resets the dirty bit of the graph.

Fig. 1 displays the code for the Flag Vector Calculation Thread. This thread involves the use of a *count* variable, which acts as the dirty bit to determine whether an edge weight has changed or not. The designation *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, then some of the edge weights have changed. A *sync* variable is used that is an index of a shared array between the Flag Thread and the Timer Thread to determine which edge has changed. After determining which edge has changed, the region of its target vertex (variable *r*) is determined, and pre-processing is performed for that specific region *r;* then, the *update time* is recorded as the time taken by the pre-processing step.

Fig. 2 involves the code for the 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. The *count* variable denotes the number of edge-weight changes that have occurred. The Timer Thread performs this operation at repeated intervals of time. This interval is determined by the *usleep*() function, which makes the thread idle for a specific amount of time. The above code uses a 2-s timer, such as *usleep* (2000000).

3.2. The Time-Dependent RSB and Elliptic Convolution of shortest path method for the shortest path computation

Many pre-processing techniques involve a large amount of computation whose results must be stored in a specific data structure for efficient point-to-point queries. However, each of these techniques involves computation on the order of hours. This approach is not suitable for networks in which the edge weights change with time. Hence, a new method is proposed to reduce the pre-processing effort and the search space and yet is feasible in addressing dynamic scenarios. The graph is partitioned using Recursive Spectral Bisection in the pre-processing phase, and later, an ellipse is constructed where the shortest path computation is performed.

3.2.1. Pre-processing phase

Pre-processing of the graph is performed using the Recursive Spectral Bisection Method. This method is a graph-partitioning strategy that is based on the degrees of the vertices. The partitions produced by the method are logical partitions. Every partition comprises vertices that have nearly the same degree. The shortest path routing is achieved by correcting the distance label of the boundary vertices of the partitions. The steps involved in the graph partitioning are given by the following:

Steps:

1. Construction of the Laplacian Matrix of the given graph

$$1_{i,j} = \begin{cases} +1, \text{edge } (i,j) \\ -\text{degree}(i) \text{ if } i = j \\ 0 \text{ otherwise} \end{cases} \tag{1}$$

```
Input : Grpah G, Edge array with cost, Edge array with flags
and Node array
Output : Regionwise updating of edges with an equivalent
assignment of flags
Begin
 While (True)
Begin
While (Count<1)
Begin
Edge e = A.get((sync++)%100); //the edge that is updated by a
TimerThread is fetched //sync is a global variable for
synchronization
int r = coord[target(3)].region;
call assignFlags(G, cost, ea, coord,r); //assign flags run on the
region r
 float updateTime = used_time(T);
count --; //job done for one change
cout << "Update Time : " << updateTime << "\n";
end
end
```

Figure 1 Pseudocode for the Flag Vector calculation thread.

```
Input: edge array and size of the array
Output: Updates the weight of a random edge at repeated
intervals of time.
Begin
number = A number from 5 to 10 is randomly chosen using
function rand_int()
e= an edge
while (true)
Begin
usleep(2000000); //2 second timer, that makes the thread idle for
an amount of time.
For the number of edges to change do
Begin
 j = rand_int(0,size-1);  // size is the size of the edge array
e = E.get(j); // get an edge from edge array
cost[e] += rand_int(-20,20);  //update cost of the edge
 A.set((sync++)%100,e); //update edges
count++; //number of edges that needs to be addressed is one
more
end
```

Figure 2 Pseudocode for the Timer Thread.

1. Finding the Second Eigen-Vector of the Laplacian Matrix (the Fiedler Vector)
2. Sorting the Fiedler Vector
3. Bisecting the graph using the Sorted Fiedler Vector

3.2.2. Shortest path computation phase

Second, the method suggested involves the construction of ellipses on the fly for a given (s, t) pair of vertices with the size of the ellipse sufficient to enclose the shortest path between (s, t).

Assuming a graph with vertices plotted according to Euclidean Distances, going with an ellipse is justified as follows:

1) The shortest path distance in the graph will be at least the Euclidean distance between the pair of vertices, as in Fig. 3.
2) The locus of points for which the sum of the distances from point P on the ellipse to s and to t is a constant for an ellipse.
3) Therefore, the shortest path would involve vertices within an elliptical boundary, such as in Fig. 4.

Steps:

1. *Translating latitudes and longitudes into algebraic co-ordinates*

To construct an ellipse, its algebraic equation must be formulated. To formulate the algebraic equation, latitudes and longitudes must be translated into (x, y) co-ordinates.

This arrangement occurs because the (x, y) co-ordinates are planar while the latitude and longitude co-ordinates represent a curved surface. To be able to accomplish this task, the curved surface must be flattened into a planar surface. This goal is accomplished as follows:

1. Assume that there is a curved line (s, t) with its latitude co-ordinates $L(s)$ and $L(t)$. The Haversine formula gives the curved line a distance from s to t. Let this distance be d, as in Fig. 5.

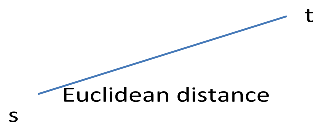

Figure 3 Euclidean distance between s and t.

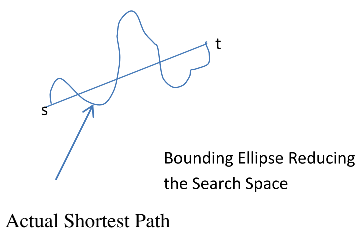

Figure 4 Shortest path within an elliptic boundary.

2. Next, fix s as the origin, i.e., the Algebraic Co-ordinate of s is 0
3. Then, the Algebraic Co-ordinate of t will be d. Thus, the coordinates are
4. $X(s) = 0$ and $X(t) = d$, as in Fig. 6.

The above procedure is also repeated for the longitude co-ordinates, and therefore, an XY plane can be established.

1. *Setting up the equation of the ellipse*

The equation of an ellipse with its center at the origin and the foci on the X-axis, as in Fig. 7, is given by

$$\frac{X^2}{a^2} + \frac{Y^2}{b^2} = 1$$

where a and b are the length of the semi-major and semi-minor axes.

$$b = a\sqrt{1 - e^2}$$

where e is the eccentricity of the ellipse. The ellipse for s, t can be oriented in any direction.Therefore, the origin must be shifted appropriately, and the ellipse is rotated so that the ellipse appears as in Fig. 8.

The modified equation of the ellipse is then

$$\frac{\left(X - \left(\frac{x_1+x_2}{2}\right)\right)^2}{a^2}\cos\theta + \frac{\left(Y - \left(\frac{y_1+y_2}{2}\right)\right)^2}{b^2}\sin\theta = 1$$

1. *Constructing the sub-graph*

Vertices that are inside or outside the ellipse can be found by substituting the (x, y) co-ordinates in the equation of the

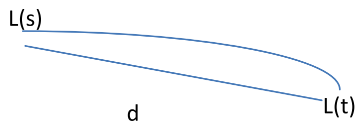

Figure 5 Curved line distance d.

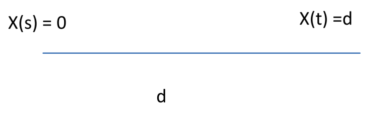

Figure 6 Translation of latitude co-ordinates to algebraic co-ordinates.

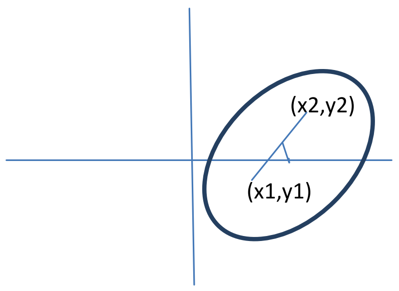

Figure 7 Ellipse construction.

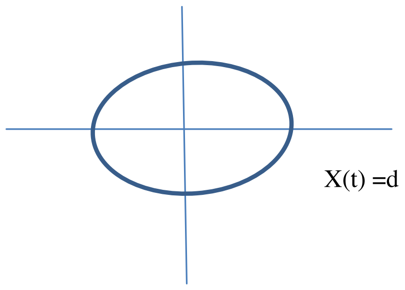

Figure 8 Ellipse rotation.

ellipse. Then, $\frac{X^2}{a^2}+\frac{Y^2}{b^2}\leqslant 1$, and the vertex lies inside the ellipse; otherwise, the vertex lies outside the ellipse. Only those edges whose end vertices are within the ellipse are considered to be a part of the sub-graph.

1. *Running Dijkstra on the sub-graph*

Dijkstra's algorithm is run for s, t on the sub-graph instead of the whole graph. The only overhead for this technique is building the sub-graph, which takes $O(m+n)$ time, where m and n are the number of edges and the number of vertices in the graph, respectively. The complexity of Dijkstra's algorithm on the sub-graph would be $O(k \log k)$, where k is the number of vertices in the sub-graph and $k \leqslant n$.

3.3. Parallel Time-Dependent RSB combined with Elliptic Convolution of the shortest path method for shortest path computation

The TD-RSB combined with the Elliptic Convolution of the shortest path method is parallelized in the precomputation phase and update phase using OpenMp parallel programming constructs. By adapting the fork and join model of OpenMp constructs, the TD-RSB and Elliptic Convolution of shortest path method is parallelized and executed in parallel.

Fig. 9 displays the code for simulating the TDSPP scenario. Both the threads, i.e., the Flag Thread and Timer Thread, are started simultaneously, and both of them run in parallel. The edges changed by the Timer Thread are appropriately updated by the Flag Thread, as explained in Figs. 1 and 2.

```
//Parallel pseudo code Executed in Main Simulating
TDSPP
#pragma omp parallel
**Begin**
#pragma omp sections
**Begin**
#pragma omp section
**Call** FlagThread(G, cost, ea, coord);
#pragma omp section
**Call** TimerThread(cost,edgeCount);
**end**
**end**
```

Figure 9 Pseudocode for parallel code simulating TDSPP.

4. Experimental results

Computation of the shortest path in dynamic networks using Time-Dependent Combined RSB and Elliptic Convolution of shortest path method and parallel Time-Dependent Combined RSB and Elliptic Convolution of shortest path method is implemented and experimented on in the following system configuration. All of the results are compiled on an Intel Core 2 Duo Desktop clocking 2.83 GHz with 4 GB RAM running on Ubuntu 10.04 Linux. The programs are written in C++ and LEDA, i.e., Library of Efficient Data types and Algorithms. LEDA is used for the graph data structures.

4.1. Dynamic test environments

The speedup techniques are implemented and tested in random, planar graphs and real-world graphs. The real-world graphs are extracted from the Tamil Nadu map OSM file. The metrics measured are the Update Time and QPL (Query Performance Loss), which determine the impact of the speedup technique with respect to ~~t~~the time-dependent scenario. The random graph can be considered for network applications or as a model of real-world networks such as the Internet, social networks or biological networks. The planar graphs can be considered in applications such as telecommunications (e.g., spanning trees) and vehicle routing (e.g., planning routes on roads without underpasses). The real-world network is constructed using a real-world data set from open street map. For all of the real road network simulations, extracts from the Tamil Nadu OSM file are used. Building the graph data structure from the OSM file involves parsing the file. First, node tags are parsed by the < node > tag, and for every parse of the tag, a new node is created in the graph and its ID is stored. Then, the < way > tags are parsed by building edges for every consecutive pair of < nd > tags. For the road networks, the nodes range from 1000 to 10,000 and the runtime, update time and pre-processing time are tabulated, and from that, the output metric query performance loss is evaluated and tabulated.

The important metrics used for the Time-Dependent Scenario are the Update Time and Query Performance Loss (QPL). The Update Time is the time taken by the speed-up technique when an edge weight changes to update the data structure so that it can be used for further point–point shortest path queries. QPL is the ratio of the time taken to solve the query after an update to the time taken to solve the query by pre-processing from scratch. QPL is a specific property of the speed-up technique and is useful to measure the suitability of the technique to the time-dependent case. The lower the QPL, the better suited the speedup technique for the time-dependent case.

It can also be stated that the metric Query Performance Loss chosen to evaluate the performance is the ratio between the average query time after the execution of the edge weight change and the time obtained for the from-scratch recomputation. This value is referred to as the Query Performance Loss (QPL).

$$\text{QPL} = \frac{(\text{Update Time} + \text{Running Time})}{(\text{Pre-processing time} + \text{Running time})}$$

4.2. Comparison of the RSB-based partitioning with the Grid partitioning technique

Here, the RSB-based speedup technique is compared with the arc-flag method. The arc-flag method (Mohring et al., 2006) was originally solved by adapting the grid partitioning strategy. In this part of the section, the grid partitioning and RSB-based partitioning results are experimented on random, planar and road networks, and the results are tabulated in Tables 1–3, respectively.

The tabulated data show the QPL results for the arc-flag method (Mohring et al., 2006) using grid partitioning strategy, the QPL results for the same arc-flag method with a Recursive Spectral Bisection partitioning strategy (Zhang et al., 2010) and the QPL results for the proposed RSB partitioning combined with the Elliptic Convolution of the shortest path method in random graphs. The proposed new technique is compared with the existing techniques of the arc-flag method using the grid partitioning strategy and an arc-flag method with Recursive Spectral Bisection partitioning strategy. The RSB partitioning reduces the Query Performance Loss because the update time is zero. The average speedup in the QPL value is 38 in RSB combined with the Elliptic Convolution of shortest path method. The speedup assumes the value of 21 in the RSB partitioning technique.

The QPL results for the proposed RSB partitioning combined with the Elliptic Convolution of the shortest path method are compared with other techniques in planar graphs. In RSB combined with the Elliptic Convolution of the shortest path method, the QPL is reduced by 8 lakh times compared with the original Grid-based arc-flag method.

The tabulated data shows the QPL results for the arc-flag method using the grid partitioning strategy, the QPL results for the same arc-flag method with the Recursive Spectral Bisection partitioning strategy and the QPL results for the proposed RSB partitioning combined with Elliptic Convolution of shortest path method in road networks. The results show that an average speedup in the QPL of 140 is achieved in the RSB combined with the Elliptic Convolution of the shortest path method.

4.3. Time-Dependent Combined RSB and Elliptic Convolution Technique

The Time-Dependent Combined RSB and Elliptic Convolution of the shortest path method are applied in dynamic networks using a time-dependent modeling approach. The modeling approach takes two types of threads, the Timer Thread and the Flag Thread, which have an impact on the output parameters: the update time and the metric QPL. The Timer Thread is used to change the edge weight at a random time to make the graph dynamic. The Flag Thread is used to assign the flag for a newly changed edge and also to compute the shortest path by using the assigned flag. The results for TD-RSB and the Elliptic Convolution of the shortest path method are recorded in Tables 4–6 for random graphs, planar graphs and road networks, respectively.

The tabulated value for the update time parameter is zero, which implies the simplicity of the pre-processing technique, and subsequently, the pre-processing time and QPL are the lowest for this technique. In this technique, the shortest path is enclosed within the ellipse. This fact reduces the search space to a greater extent. Hence, the running time is also reduced. The formula for QPL has the running time and update time in the numerator. Because the update time is zero in the RSB-based speedup technique, the QPL depends on the running time and the pre-processing time.

From Table 4–6 and Fig. 10, it is evident that real and planar graphs exhibit the same performance with the least QPL among all of the techniques. The node to edge ratio ranges from 1:1.5 to 1:1 in planar and road networks, respectively. The reduction in the number of edges reduces the pre-processing time, which has a significant impact on its QPL. The node to edge ratio in the random networks is 1:2, which doubles the number of edges, and correspondingly, the pre-processing time is also increased. The performance is better with planar graphs compared to random graphs.

Table 1 Comparison of RSB-based Speedup Techniques with the grid partitioning-based arcflag technique on random graphs.

Nodes	Edges	Grid-QPL	RSB-QPL	RSB-EL QPL
1000	2020	0.77994	0.02947	9.50E-07
2000	4041	0.74776	0.02913	1.03E-06
3000	6049	0.75744	0.03382	1.52E-06
4000	8078	0.75124	0.03759	7.20E-07
5000	10,099	0.74879	0.04337	9.38E-07
6000	12,112	0.75937	0.03306	0.115385
7000	14,112	0.75455	0.03492	0.0384622
8000	16,160	0.76278	0.04322	6.24E-07
9000	18,186	0.80406	0.03192	1.13E-06
10,000	20,196	0.75715	0.0355	0.0454554

Table 2 Comparison of RSB-based Speedup Techniques with the grid partitioning-based Arc-flag technique on planar graphs.

Nodes	Edges	Grid-QPL	RSB-QPL	RSB-EL QPL
1000	1507	0.73333	0.03824	1.51991e-06
2000	3204	0.74602	0.03221	7.07804e-07
3000	4914	0.75232	0.03534	7.69702e-07
4000	6667	0.75243	0.0241	7.96065e-07
5000	8506	0.74447	0.04288	9.61123e-07
6000	10,283	0.74632	0.04238	6.86517e-07
7000	12,061	0.75029	0.02393	8.01952e-07
8000	13,872	0.74496	0.03093	7.03478e-07
9000	15,674	0.75549	0.01888	1.01944e-06
10,000	17,498	0.74634	0.03858	1.08996e-06

Table 3 Comparison of RSB-based Speedup Techniques with the grid partitioning based arc-flag technique on road networks.

Nodes	Edges	Grid-QPL	RSB-QPL	RSB-EL QPL
1000	1000	0.0004	0.04	2.00606e-06
2000	2001	0.00101	0.0466	1.96695e-06
3000	3003	0.00092	0.05598	2.64246e-06
4000	4037	0.00059	0.03209	2.97104e-06
5000	5063	0.00052	0.03829	3.35039e-06
6000	6066	0.00037	0.02718	3.03982e-06
7000	7079	0.00045	0.05068	3.53479e-06
8000	8093	0.00026	0.03222	4.05309e-06
9000	9126	0.00024	0.04781	5.12211e-06
10,000	10,132	0.0002	0.02861	6.55302e-06

Table 4 Time-Dependent RSB and enclosing ellipse technique on random graphs.

Nodes	Edges	Pre-Processing Time (ms)	Update Time (ms)	QPL
1000	2020	0.0004	0	9.50E-07
2000	4041	0.0006	0	1.03E-06
3000	6049	0.0008	0	1.52E-06
4000	8078	0.0014	0	7.20E-07
5000	10,099	0.0021	0	9.38E-07
6000	12,112	0.0023	0	0.115385
7000	14,112	0.0025	0	0.0384622
8000	16,160	0.0034	0	6.24E-07
9000	18,186	0.0033	0	1.13E-06
10,000	20,196	0.0042	0	0.0454554
	Average	0.0021	0	0.01993095

Table 5 Time-Dependent RSB and enclosing ellipse technique on planar graphs.

Nodes	Edges	Pre-Processing Time (ms)	Update Time (ms)	QPL
1000	1507	0.0003	0	1.51991e-06
2000	3204	0.0009	0	7.07804e-07
3000	4914	0.0013	0	7.69702e-07
4000	6667	0.0013	0	7.96065e-07
5000	8506	0.0017	0	9.61123e-07
6000	10,283	0.0028	0	6.86517e-07
7000	12,061	0.0022	0	8.01952e-07
8000	13,872	0.0031	0	7.03478e-07
9000	15,674	0.0029	0	1.01944e-06
10,000	17,498	0.0041	0	1.08996e-06
	Average	0.00206	0	1.543E-06

Table 6 Time-Dependent Combined RSB and enclosing ellipse technique on road networks.

Nodes	Edges	Pre-Processing Time (ms)	Update Time (ms)	QPL
1000	1000	0.0004	0	2.00606e-06
2000	2001	0.0007	0	1.96695e-06
3000	3003	0.0009	0	2.64246e-06
4000	4037	0.0013	0	2.97104e-06
5000	5063	0.00190001	0	3.35039e-06
6000	6066	0.00220001	0	3.03982e-06
7000	7079	0.00230001	0	3.53479e-06
8000	8093	0.00260001	0	4.05309e-06
9000	9126	0.00310001	0	5.12211e-06
10,000	10,132	0.00350002	0	6.55302e-06
	Average	0.00189001	0	3.524E-06

The low value of QPL indicates that the Time-Dependent RSB and the Elliptic Convolution of the shortest path methods are well suited for planar graphs and road networks. This relationship occurs because planar graphs and road networks have very few edges. Because the number of edges is smaller, the pre-processing time is also reduced.

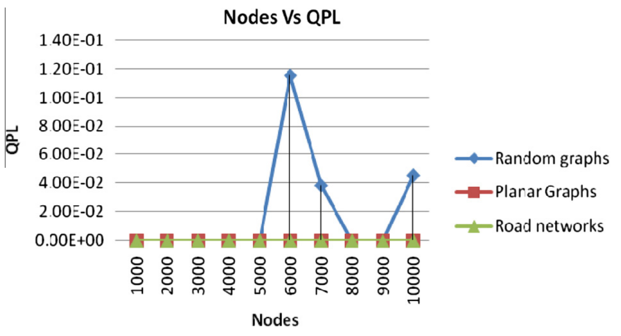

Figure 10 Performance of TD-RSB and Elliptic Convolution.

Table 7 Parallel Time-Dependent RSB and ellipse convolution technique on random graphs.

Nodes	Edges	Pre-Processing Time (ms)	Update Time (ms)	QPL
1000	2020	0.0003	0	0.250001
2000	4041	0.0007	0	8.84489e-07
3000	6049	0.000800001	0	1.14273e-06
4000	8078	0.0013	0	9.14701e-07
5000	10,099	0.0017	0	8.82235e-07
6000	12,112	0.002	0	0.0476198
7000	14,112	0.0029	0	0.0333339
8000	16,160	0.0032	0	0.0303037
9000	18,186	0.0038	0	0.0731715
10,000	20,196	0.0037	0	8.1473e-07
	Average	0.00204	0	0.08688598

4.4. Parallel Time-Dependent RSB and Elliptic Convolution Technique

The objective of parallelization is to improve the performance of the proposed technique. The incorporation of parallelization in a dynamic scenario will improve the speedup of the system. The operations of the Timer Thread and Flag Thread are implemented using the multithreaded approach in this technique. The two OpenMp constructs used for implementing this technique are *parallel for* and *parallel sections*.

The results of parallel TD-Combined RSB and Elliptic Convolution of the shortest path method are tabulated in Table 7–9 for random graphs, planar graphs and road networks, respectively.

In earlier discussions, the QPL values of the TD-Combined RSB and Elliptic Convolution of the shortest path method are compared with existing speedup techniques, namely, the Arc and RSB techniques. It is concluded that the QPL values are better in the TD-Combined RSB and Elliptic Convolution of the Shortest Path method. Hence, the results of the parallel technique are presented, and the parallel speedup is calculated.

The number of edges generated is doubled with respect to the nodes in the random graphs. This relationship is reflected in the Query Performance Loss. The average speedup for QPL is 0.23, which is much less and allows an increase in the number of nodes of more than 10,000. The pre-processing time is almost the same in the parallel TD-RSB and Elliptic Convolution of the shortest path method compared with the sequential approach.

Table 8 Parallel Time-Dependent RSB and enclosing ellipse technique on planar graphs.

Nodes	Edges	Pre-Processing Time (ms)	Update Time (ms)	QPL
1000	1507	0.0004	0	1.2964e-06
2000	3204	0.0009	0	7.39262e-07
3000	4914	0.001	0	1.23903e-06
4000	6667	0.0016	0	6.81727e-07
5000	8506	0.002	0	9.49203e-07
6000	10,283	0.0019	0	9.1446e-07
7000	12,061	0.0024	0	7.23326e-07
8000	13,887	0.0034	0	7.37168e-07
9000	15,674	0.0035	0	9.78366e-07
10,000	17,535	0.0035	0	9.07692e-07
	Average	0.00206	0	9.17E-07

Table 9 Parallel Time-Dependent RSB and enclosing ellipse technique on road networks.

Nodes	Edges	Pre-Processing Time (ms)	Update Time (ms)	QPL
1000	1000	0.0005	0	1.4171e-06
2000	2001	0.000700001	0	1.87754e-06
3000	3003	0.0013	0	1.68268e-06
4000	4037	0.0011	0	3.25114e-06
5000	5063	0.00180001	0	3.53652e-06
6000	6066	0.00200001	0	3.30804e-06
7000	7079	0.00250001	0	3.16618e-06
8000	8093	0.00270001	0	4.23852e-06
9000	9126	0.00320001	0	4.78323e-06
10,000	10,132	0.00360002	0	5.32E-006
	Average	0.001940007	0	3.5239E-06

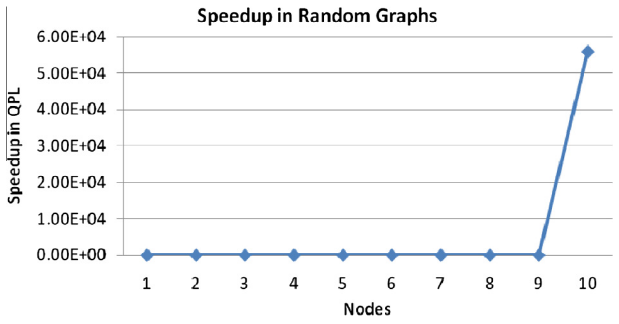

Figure 11 Speedup Plots of the Query Performance Loss of the parallel Time-Dependent Combined RSB and the enclosing ellipse technique on random graphs.

From Fig. 11, it is evident that parallelization gives a maximum increase in speedup of 55,800 for a node count of 10,000. The speedup is expected to increase for more than 10,000 nodes. Compared to random graphs, planar graphs show an improvement in the QPL. One reason for this improvement is that there is a linear increase in the number of edges for the nodes considered.

The QPL values are reduced in the parallel TD-Combined RSB and Elliptic Convolution of the shortest path method

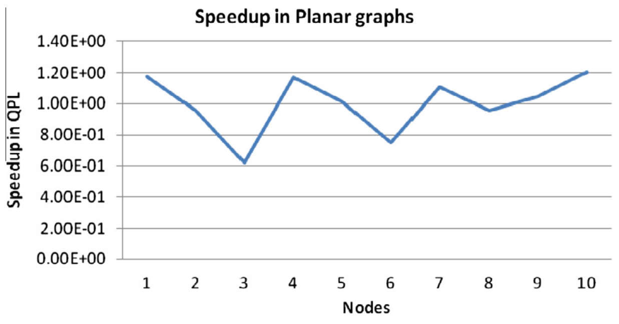

Figure 12 Speedup Plots of the Query Performance loss of parallel Time-Dependent RSB and enclosing ellipse technique on planar graphs.

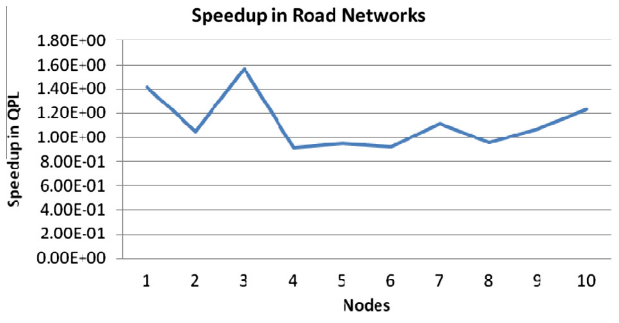

Figure 13 Speedup Plots of the Query Performance Loss of parallel Time-Dependent RSB and enclosing ellipse technique on road networks.

because the parallelized operations of the Flag Thread and Timer Thread for the reduced edges dominate the sequential method.

The average speedup for QPL for planar graphs is 1.68, which is better than that of random graphs when the number of nodes is in the range of 1000–10,000. It is obvious from Figs. 11 and 12 that the impact of parallelization is very good when there are more than 10,000 nodes. Planar graphs have a smaller number of edges. Hence, the pre-processing of the edges will take less time, which will reduce the QPL.

The results of parallel TD-RSB and enclosing ellipse techniques on road networks are tabulated in Table 9. This table shows a much smaller improvement in the speedup for QPL. The pre-processing time obtained due to a parallelized approach is also much less.

From Fig. 13, it is evident that the average speedup for QPL is 1.000032851, which is marginally greater than one and is better than random graphs. Here, the edges considered for the Shortest Path computation play a very important role.

It can also be concluded that the technique of parallelization shows improvement in terms of the pre-processing time and QPL. Usually, the pre-processing-based speedup techniques show improvement in the query performance.

5. Conclusions

The shortest path computation for the Time-Dependent Shortest Path Problem (TDSPP) is solved using a Speedup Technique called the Recursive Spectral Bisection Combined with

Elliptic Convolution of shortest path method. The same method is parallelized using OpenMP parallel programming constructs, and the results are recorded and compared with its sequential counterpart. The TD-Recursive Spectral Bisection combined with Elliptic Convolution of shortest path method shows a reduced QPL value in the planar type of networks and road networks. Then, the results are compared with parallel TD-Combined RSB and Elliptic Convolution of shortest path method. The results show that the performance of parallelization is realized in the planar type of networks rather than random and road networks, from which it can be concluded that the number of edges considered for the shortest path computation plays an important role.

Furthermore, the combined Recursive Spectral Bisection with Elliptic Convolution of shortest path method can be combined with other speedup techniques. The corresponding parallelized technique can be applied to new real-world applications that use multi-core processors.

Acknowledgments

The authors wish to thank the students S. Sarat Chandra, Sarath Subramaniam and Vivek Chandra, who are students of the Pondicherry Engineering College and who helped to conduct the experiments.

References

Berrettini, E., D'Angelo, G., Delling, D., 2009. Dynamic arc-flag in road network. In: 9th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems, ATMOS 2009. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, Germany.

D'Angelo, Gianlorenzo, D'Emidio, Mattia, Frigioni, Daniele, Vitale, Camillo, 2012. Fully dynamic maintenance of arc-flags in road networks. 11th International Symposium-SEA 2012, 7–9 June, 2012, LNCS 7276 147, 135.

D'Angelo, Gianlorenzo, Frigioni, Daniele, Vitale, Camillo, 2011. Dynamic Arc-Flags in Road Networks. In the proceedings of SEA 2011, Lecture Notes in Computer Science 6630/2011 (99), 88.

Delling, Daniel, Nannicini, Giacomo, 2008. Bidirectional Core-Based Routing in Dynamic Time-Dependent Road Networks, ISAAC 2008, LNCS 5369, 824(2008) 813, Springer-Verlag Berlin Heidelberg.

Delling, Daniel, Wagner, Dorothea, 2007. Landmark-based routing in dynamic graphs, experimental algorithms. Lecture Notes in Computer Science 4525 (65), 52.

Ding, Bolin, Yu, Jeffrey Xu, Qin, Lu, 2008. Finding Time-Dependent Shortest Paths over Large Graphs. EDBT'08, Nantes, France.

Ehmke, Jan Fabian, Mattfeld, Dirk Christian, 2010. Data allocation and application for time-dependent vehicle routing in city logistics, European transport. Trasporti Europein 46 (35), 24.

Li, Yinzhen, He, Ruichun, Zhang, Zhongfu, Guo, Yaohuang, 2005. Models and algorithms for shortest paths in a time dependent network. International Symposium on OR and its Applications 328, 319.

Mohring, R.H., Schilling, H., Schutz, B., Wagner, D., Willhalm, T., 2006. Partitioning graphs to speed up Dijkstra's algorithm. ACM Journal of Experimental Algorithmics 11, 1–29.

Sibai, Fadi N., 2013. Performance modeling and analysis of parallel Gaussian elimination on multi-core computers. Journal of King Saud University – Computer and Information Sciences. Elsevier, Vol. 26, pp. 41–54.

Wagner, D., Willhalm, 2003. Geometric Speed-Up Techniques for Finding Shortest Paths in Large Sparse Graphs. In: Di Battista, G., Zwick, U. (Eds.), Proc. Algorithms ESA 2003: 11th Annual European Symposium on Algorithms. Volume 2832 of LNCS., Springer (2003), 787, p. 776.

Wagner, Dorothea, Willhalm, Thomas 2007. Speed-up Techniques for Shortest-path Computations, IST priority 6th FP, University at Karlsruhe, Germany.

Zhang, Lin, Yang, Zhaosheng, Jia, Hongmei, Wang, Bin, Chen, Guang, 2010. Test and Implement of a Parallel Shortest Path Calculation System for Traffic Network, ICICA 2010, Part I, CCIS 105, 288(2010) 282, © Springer-Verlag Berlin Heidelberg.