# Re-optimizing the Performance of Shortest Path Queries Using Parallelized Combining Speedup Technique based on Bidirectional Arc flags and Multilevel Approach

R. Kalpana[#1] P. Thambidurai[*2]
[#]*Department of Computer Science & Engineering, Pondicherry Engineering College*
*Puducherry, India*
[1]rkalpana @pec.edu
[*]*Perunthalaivar Kamarajar Institute of Engineering & Technology*
*Karaikal, Puducherry, India*

***Abstract:*** - Globally shortest path problems have increasing demand due to voluminous datasets in applications like roadmaps, web search engines, mobile data sets, etc., Computing shortest path between nodes in a given directed graph is a very common problem. Among the various shortest path algorithms, Dijkstra's shortest path algorithm [1] is said to have better performance with regard to run time than the other algorithms. The output of Dijkstra's shortest path algorithm can be improved with speedup techniques. In this paper a new combined speedup technique based on three speedup techniques were combined and each technique is parallelised individually and the performance of the combination is measured with respect to pre-processing time, runtime and number of nodes visited in random graphs, planar graphs and real world data sets.

***Key-Words:*** **-** Bidirectional Arcflags, Multilevel method, Multilevel Arcflags, Parallelized Multilevel Arcflags**.**

## 1 Introduction

In general many applications require shortest path queries to solve the problems. Some of the applications are railway networks [2],[3], roadmaps [3] , web search engines [3] , mobile applications, etc., The need for shortest path queries have extended due to online applications, where the search time is reduced due to shortest path queries. Shortest path problems are classically solved under Greedy procedures. The commonly known shortest path algorithms of greedy are Dijkstra's Algorithm, Bellmann-Ford Algorithm, Floyd-Warshall's Algorithm, etc., Dijkstra's algorithm [1],[2] is the standard algorithm which computes shortest path in directed graphs with non negative edge lengths. Dijkstra's algorithm with Fibonacci heaps is the fastest algorithm for the general case of arbitrary nonnegative edge lengths. The performance of Dijkstra's algorithm can be extended using basic speedup techniques like bidirectional search, goal directed search, shortest path containers, multilevel approach, reach based method, arc flag method, etc., to find the shortest path in optimal time. The basic speedup techniques [3],[4] were combined in different flavors and their performance were improved. These basic speedup techniques and combined speedup techniques cannot be always guaranteed to prove to be faster than the original Dijkstra's algorithm. However it can be empirically shown that they certainly improve the speedup of the applications where we use many real data sets like roadmaps [5], railway networks [2],[6] and timetable information systems [6], etc.,.

The shortest path problem has two phases of implementation for applications where there is a need for voluminous data sets. They are pre-processing phase and shortest path computation phase. Pre-processing techniques were identified to make the applications to work fast. It makes to work fast in very large networks, where there is a need for many 1 to n shortest path computations. The speed up factor is found to be high in techniques where pre-processing the network is done at the design phase of the network itself [2],[3]. In the shortest path computation phase, actual speedup techniques integrated with Dijkstra's algorithm works to give the result in optimal time. Hence the output of the system can be measured with output parameters like pre-processing time, runtime(shortest path computation time), Number of nodes visited during shortest path computation, etc.,

In this paper, a combination of the Bidirectional Arcflags (Goal-directed search) and Multilevel technique has been considered to improve the speedup in terms of run time and vertex visit count in various graph types such as random graphs, planar graphs and real world data sets(map of Tamilnadu). Pre-processing time of the system

also reduced due to parallelism and thereby the speedup of the system got improved.

## 2 Related Work

### 2.1 Combining Speedup techniques

A detailed view of the existing combinations of various speedup technique are discussed in [3], [4]. The speedup techniques includes Bidirectional search [3], [4], [7], goal directed search [8],[9], Hierarchical approaches [10],[11], Reach Highway hierarchies [5] and Transit node routing [12] and goal directed technique include ALT [8] and Arcflags [13]. Especially in arc flag approach [13] various graph partitioning methods adapted improves the capacity of pre-processing.

The combinations Goal-Directed Search and Multilevel Approach, Goal-Directed Search and Shortest-Path Containers[14], Bidirectional Search and Multilevel Approach [15], Bidirectional Search and Shortest-Path Containers, Multilevel Approach and Shortest-Path Containers with its speedup with respect to running time and vertices visited were analysed in [3]. Combination of reach with landmark based A* search (ALT algorithm) [8]. Another variation of this combination is to store the landmark distances of nodes with high reach values and this results in low memory consumption. HH* combines Highway Hierarchies approaches [10,[11] with landmark based A* search [8]. Here the landmarks are not chosen from the original graph, but for some level k of the HH (highway hierarchy), which reduces the pre-processing time and memory consumption. SHARC combines Highway hierarchies [10] with Arc-flags [13] and produces a fast unidirectional query algorithm, which is advantageous in scenarios where bidirectional search is prohibitive, like road networks. Combining hierarchical approaches with goal directed search [16], [17]have good results in real world problems.

In Highway Node routing **r**esults of shared-memory parallel variants of the multi-level overlay graph construction necessary for HNR are discussed in [18]. A high number of updates per time is desirable to keep the replies to the shortest path queries as up-to-date as possible. On a modern processor, the repeated precomputation step for HNR takes roughly two minutes.

The parallel programming constructs are applied to Bidirectional search [19], Landmark technique[20], and Bidirectional arc flags[21] using OpenMP [22], which proves to give better results in speedup factor in random and planar and real world graphs.

Shared memory parallel programming[18] constructs of OpenMP ([22] are considered in the preprocessing phase of graphs. Using the work sharing constructs, the time taken for preprocessing can be reduced.

### 2.2 Parallel Programming

Shared memory parallel programming adapts the principles of Amdahl's law. Amdahl's law [23] states that if $T1$ denotes the execution time of an application on 1 processor, then in an ideal situation, the execution time on $P$ processors should be $T1/P$. If $TP$ denotes the execution time on $P$ processors, then the ratio

$$S = T_1 / T_P$$

is referred to as the parallel speedup and is a measure for the success of the parallelization. However, a number of obstacles usually have to be overcome before perfect speedup is achievable. Virtually all programs contain some regions that are suitable for parallelization and other regions that are not. By using an increasing number of processors, the time spent in the parallelized parts of the program is reduced, but the sequential section remains the same. Eventually the execution time is completely dominated by the time taken to compute the sequential portion, which puts an upper limit on the expected speedup. This effect, known as *Amdahl's law*, can be formulated as

$$S = \frac{1}{(f_{par} / P + (1 - f_{par}))}$$

Where $f_{par}$ is the parallel fraction of the code and $P$ is the number of processors. In the ideal case when all of the code runs in parallel, $f_{par} = 1$, the expected speedup is equal to the number of processors.

### 2.3 Parallelized Bidirectional dijkstra's algorithm with arc flag

DEFINITION - **BIDIRECTIONAL ARC FLAGS VECTOR**

Let G =(V,E) be a weighted graph together with a weight function l then for each arc $e$ belonging to E the nodes in the regions $r_i$, which are associated with the true entries of the arc flag vector of e, constitute a *consistent bidirectional arc flag vector.*

Let G= (V, E) be a weighted graph together with the weight function l. We call a set of nodes C sub set of V, as bidirectional arc flag vector [13]. A bidirectional arc flag vector C associated with an arc from u to v is called consistent if for all shortest path from u to t that start with the arc from u to v, the target node t is in C. Similarly if for all shortest path from t to u that start with the arc from t to s, the target node u is in C.

Consider the shortest path p from s to t that is found by a Dijkstra's algorithm. If for all arcs, e belonging to E, the target node t is in the

bidirectional arc flag vector C of e, then the path p will also be found by bidirectional Dijkstra's algorithm with Arc Flags. Similarly, if the shortest path from t to s is found to be $P^1$ by Dijkstra's algorithm, the same will also be found by bidirectional Dijkstra's algorithm with arc-flags. This is because arc-flags do not change the order in which the arcs are processed. A sub path of a shortest path is again a shortest path, so for all arcs from u to v belonging to P, the sub path of P from u to t is a shortest path. Hence by definition of the consistent arc flag vector [13] t belongs to bidirectional arc flag vector C. The above procedure when done simultaneously in the forward and backward directions has been proved to lower the run time by a reasonable amount.

The parallelizing bidirectional Dijkstra algorithm includes two phases of implementation: preprocessing of arc flags and shortest path computation. The preprocessing phase of Arcflags deals with calculating the arc-flag entries for all arcs [13]. The arc flag preprocessing phase is outlined in Algorithm 1. This can be achieved by computing a shortest path tree from every arc *a* to all nodes in the graph—a one-to-all shortest-path computation from the head node of arc *a*. The computation is done by a standard algorithm of Dijkstra, which stops when all nodes are permanently marked. During this computation, if a node *v* is settled, the arc-flag entry $f_a(r(v))$ is set to true for the region $r(v)$ containing node *v*.

A generalization of a partition-based arc labeling technique that is referred to as the *arc-flag approach* [13] in combination with bidirectional method is discussed here. The basic idea of the arcflag method is to use a simple rectangular geographic partition. The arc-flag approach divides the graph into regions and gathers information for each arc on whether this arc is on a shortest path into a given region. In this experimental setup, the graph is divided into a 6x6 grid. For each arc this information is stored in a vector. The vector contains a flag for each region of the graph, indicating whether this arc is on a shortest path into that particular region. The vector is called the *arcflag vector*[13] and the entries in the arc-flag vector are called the *arc-flags*. The size of each vector is determined by the number of regions and the number of vectors is determined by the number of arcs. Arc-flags are used in the Dijkstra's shortest path computation to avoid exploring unnecessary paths. When this technique is combined with Bidirectional method, it improves the speedup of shortest path queries especially in real world graphs.

### 2.4 Preprocessing for Multilevel method

An overlay graph of a given graph $G = (V, E)$ on a subset $S$ *of* $V$ is a graph with vertex set $S$ and edges corresponding to shortest paths in $G$. In particular, we consider variations of the multilevel overlay graph, a method to speedup exact single-pair shortest path computation. We restrict ourselves to overlay graphs preserving shortest path lengths. With the multilevel approach, one or more levels of overlay graphs which inherit shortest-path lengths from the base graph are constructed. Then a shortest-path computation takes place in a graph consisting basically of one of the overlay graphs and some additional edges. Procedure to generate overlay graph is given below.

Procedure **min-overlay**($G$, $l$, $S$)

For each vertex $u \in S$, run Dijkstra's algorithm on $G$ with pairs $(l_e, \alpha_e)$ as edge weights, where $\alpha_e := -1$ if the tail of edge $e$ belongs to $S \setminus \{u\}$, and $\alpha_e := 0$ otherwise. Addition is done pairwise, and the order is lexicographic. The result of Dijkstra's algorithm are distance labels $(l_v, \alpha_v)$ at the vertices, where $(l_u, \alpha_u) := (0, 0)$ in the beginning. For each $v \in S \setminus \{u\}$ we introduce an edge $(u, v)$ in $E'$ with length $l_v$ if and only if $\alpha_v = 0$.

By iteratively applying the min-overlay procedure with a sequence of subsets $S_1$, a subset of $S_2$, a subset of $S_3$ . . . a subset of $S_l$ of $V$, we obtain a hierarchy $G_i = (S_i , E_i)$ of shortest-path overlay graphs (for some $l \geq 1$). Together with $G_0 = (V_0, E_0) := G$, we call this collection of shortest-path overlay graphs, also denoted by $M(G; S_1, \ldots, S_l)$, a *basic multilevel graph* of $G$ with $l + 1$ levels.

## 3 Modified Dijkstra's algorithm with Multilevel Bi-arc flags

### 3.1 Combining Bidirectional Arc flags with Multilevel Approach

Given a graph and a subset of its vertices, an overlay graph [15] describes a topology defined on this subset, where edges correspond to paths in the underlying graph. With the multilevel approach, one or more levels of overlay graphs which inherit shortest-path lengths from the base graph are constructed. Then shortest-path computation takes place in a graph consisting basically of one of the overlay graphs and some additional edges.

A generalization of a partition-based arc labeling technique that is referred to as the *arc-flag approach [13]* is applied to the graph obtained as a result of multilevel preprocessing. The basic idea of the arcflag method is to use a simple rectangular geographic partition. The arc-flag approach divides the graph into regions and gathers information for each arc on whether this

arc is on a shortest path into a given region. In this experimental setup, the graph is divided into a 6x6 matrix. For each arc this information is stored in a vector. The vector contains a flag for each region of the graph, indicating whether this arc is on a shortest path into that particular region. The vector is called the *arcflag vector* and the entries in the arc-flag vector are called the *arc-flags*. The size of each vector is determined by the number of regions and the number of vectors is determined by the number of arcs. Arc-flags are used in the Dijkstra computation to avoid exploring unnecessary paths.

The given graph is preprocessed using Multilevel technique and Arcflag method. During the shortest path computation phase, the edge under consideration is checked if it leads to the level of target node or not. If yes, then the arcflag vector of that particular edge is considered for further shortest path computation. When this technique is applied to real world datasets, results have been proved to improve the speedup of the system.

The running time of Dijkstra's algorithm is $O(n\log n)$ time for sparse graphs, the overall running time is $O(n^2 \log n)$ plus the time to pre-process the graphs. The pre-processing time is dominated by the time needed to compute $m$ times a shortest-path tree, which can be done in $O(m + n\log n)$ time each. The resulting time complexity of the overall pre-processing at each level is, therefore, $O(m(m+ n + n\log n))$. Here, two levels are considered at the pre-processing phase. If $l$ is the number of levels, it will be $l$ times the overall pre-processing. If bidirectional search is adapted here the pre-processing time will be reduced by half as the searches move from forward and reverse direction.

The both searches expand a tree with branching factor $b$, and the distance from start to target is $d$, each of the two searches has complexity $O(b^{d/2})$, and the sum of these two search times is much less than the $O(b^d)$ complexity that would result from a single search from the starting node to the target using multilevel bidirectional arc flags.

**Pseudocode 1.** Modified Dijkstra's algorithm with Bidirectional Multilevel Arcflags

**Input**: directed graph G: = (V, A), nonnegative length $l_a$ for all a ∈ A,
Start and target nodes s, t ∈ V.
**Output**: shortest path from s to t.
**1 begin**
**2** TargetRegion:= region number of t; //coarse partition
**3** SubTargetRegion: = subregion number of t; //fine partition
**4** level(s) := level of start node s;
**5** level(t) := level of target node t;
**6** Distance(s):=0;
**7** Queue.insert(s,0);
**8** current_level := level(s);
**9** **while not** Queue.empty **do**
**10** v := Queue.extractMin;
**11** **for** all outgoing arcs (u,v) **do**
**12** **if** level(u)!=current_level
**13** **continue;**
**14** current_level := level(u);
**15** **if not** ArcFlagVectorFirstLevel [(u, v), TargetRegion] **then**
**16** continue;
**17** **if** (u,v) ∈ TargetRegion **then**
**18** **if not** ArcFlagVectorSecondLevel[(u,v), SubTargetRegion] **Then**
**continue**;
**21** **if** distance(u) ≤ distance(v) + $l_{(v,u)}$ **then**
**22** **continue**;
**23** distance(u)= distance(v) + $l_{(v,u)}$;
**if** u does not belong to Queue **then**
**25** Queue.insert(u);
**26** **else**
**27** Queue.decreaseKey(u);
**28 end**
The above procedure when done simultaneously in the forward and backward directions (from step 9 to 27) has been proved to lower the running time by a considerable amount.

### 3.2 Parallelizing the Combining Bidirectional Arc flags with Multilevel Approach

The operation that lends itself to parallelization is the updation of distance values, for the neighbours of a node which is marked permanent for all the outgoing arcs of a particular region and level. It is to be noted that another possibility for parallelism is to run both the forward and reverse variants of the algorithm simultaneously as independent threads of the search process with appropriate synchronization constructs for the shared memory access.

The code segments which can be parallelized are embedded in the following work sharing constructs.
#pragma omp for (Used for sharing iterations in a loop)
#pragma omp sections (Specify different work for each thread individually)

**Pseudocode 2.** Parallelized Modified Dijkstra's algorithm with Bidirectional Multilevel Arcflags

```
Input: directed graph G: = (V, A), nonnegative length l_a for all a ∈ A,
        Start and target nodes s, t ∈ V.
Output: shortest path from s to t.
1  begin
2     TargetRegion:= region number of t; //coarse partition
3     SubTargetRegion: = subregion number of t; //fine partition
4     level(s) := level of start node s;
5     level(t) := level of target node t;
6     Distance(s):=0;
7     Queue.insert(s,0);
8     current_level := level(s);
9      while not Queue.empty do
10         v :=  Queue.extractMin;
11         #pragma omp parallel sections
           {
12         # pragma omp section
           {
13         for all outgoing arcs (u,v) do
14            if level(u)!=current_level
15                  continue;
16            current_level := level(u);
17            if not ArcFlagVectorFirstLevel [(u, v), TargetRegion] then
18                  continue;
19            if (u,v) ∈ TargetRegion then
20                  if not ArcFlagVectorSecondLevel[(u,v), SubTargetRegion]
                    then
                       continue;
21            if distance(u) ≤ distance(v) + l_(v,u) then
22                  continue;
23            distance(u)= distance(v) + l_(v,u);
              if u does not belong to Queue then
25                  Queue.insert(u);
26            else
27                  Queue.decreaseKey(u);
                    }// end of parallel setion
28   end
```

The above procedure when done simultaneously in the forward and backward directions (from step 9 to 27) has been proved to lower the running time and number of vertices visited by a considerable amount.

### 3.3 Parallelizing the pre-processing phase of Speedup Techniques

Using OpenMP, preprocessing phase of Arcflags and Multilevel technique were parallelized and the resulting technique showed improvements in the running time and number of vertices visited when applied to random graphs, planar graphs and real world data sets. The segment of the code which can be parallelized in the speedup technique of arc flags method is highlighted in Pseudocode 3.

Pseudocode 3, **Parallelizing the preprocessing phase**

```
#pragma omp parallel sections
        {
      # pragma omp section
        {
      for all nodes in the graph do
              # pragma omp section
              {
              for i:=1 to 36 do
                region[i];=false
                r:=region_y*6 +region_x*6
                region[r]:=true //set the regions
reachable from the given node
              }
      end
} // end of the parallel section
```

As the arc flag approach resides in the levels of the search process the parallelism works for arc flag is activated at each level. In the bidirectional search, parallelism (R.Kalpana et al, 2010) is incorporated as such in the forward and reverse variants of the algorithm simultaneously as independent threads of the search process with appropriate synchronization constructs for the shared memory access.

## 4. EXPERIMENTAL SETUP

Implementation of the proposed combination was tested on a PC with AMD Athlon X2 Dual Core processor (2.1 Ghz) with 4 GB RAM running Ubuntu 9.04. Library of Efficient Date types and Algorithms (LEDA) (Algorithmic Solutions Software GmbH, 1995) was used for easy implementation of various data types such as graphs, lists, priority queues, arrays, etc..

Important metrics for evaluation of the techniques like speedup based on run time and the number of vertices visited during shortest path computation were considered. The proposed technique of combining Bidirectional Arcflags and Multilevel technique was also implemented and experimented on random and planar graphs with node count ranging from 100 to 1000 and also for a few real world data sets (Map of TamilNadu) and the results analysed. Road Map of TamilNadu was considered for testing. The first data set consisted of 17 nodes and 36 edges. The second data set consisted of 26 nodes and 62 edges. The third data set consisted of 35 nodes and 82 edges. The fourth data set consisted of 63 nodes and 146 edges. The data set consisted of most of the cities of South India.

TABLE I

COMPARISON OF SPEEDUP WITH RESPECT TO RUN TIME

| | Speedup for planar graph | Speedup for random graph | Speedup for real world data set |
|---|---|---|---|
| Arcflags | 2.33 | 0.93 | 0.000000026 |
| Bidirectional | 1.19 | 1.39 | 0.25 |
| Bidirectional Arcflags | 1 | 0.79 | 0.0000031 |
| Parallel Arcflags | 1.2 | 0.73 | 0.000519 |
| Parallel Bidirectional | 0.69 | 1.32 | 0.0909 |
| Parallel Bidirectional Arcflags | 1 | 1.08 | 1.0000019 |
| Multilevel | 0.00001223 | 0.00001862 | 2.2436484 |
| Parallel multilevel | 0.99997 | 1.24932 | 2.8440908 |
| Multilevel Arcflags | 0.816645 | 0.698968 | 2.16999 |
| Parallel Multilevel Arcflags | 0.658067 | 0.846308 | 2.685 |
| Bidirectional Multilevel Arcflags | 0.47441 | 1.4364802 | 2.700531 |
| Parallel Bidirectional Multilevel Arcflags | 0.463149 | 1.33775 | 2.91242 |

Depending on the source and target nodes, the graph is divided into various levels and the shortest path computation is done. On an average a speedup of 2.91 with respect to run time and a speedup of 3.2 with respect to vertex visit count were obtained by Parallel Multilevel Bidirectional Arcflags.

The Table I shows the comparison of run time for the various speedup techniques in random, planar and real world graphs. All the techniques

work moderately well in random graphs. Arc flags work very well in planar graphs. The performance of multilevel approach combined with other speedup techniques always work well in real world graphs. Exploiting parallelism using multithreaded programming improves the speedup better in most of the combinations. The chart demonstrating the same is shown in Fig. 1.

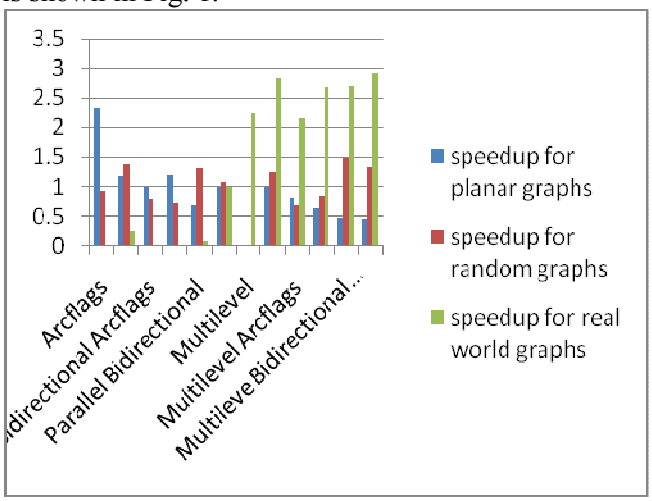


Figure 1.Comparison of speed up with respect to run time.

The Table II shows the comparison of vertex visit count for the various speedup techniques in random, planar and real world graphs. The output of the system will not get worsen in all types of graphs, whenever vertex count is considered as output metric. Here the results are better than the previous metric i.e., speedup with respect to runtime. Similar to the previous case it gives better result when parallelism is incorporated. The chart demonstrating the same is shown in Fig. 2.

The speedup techniques presented above worked well for a specific type of graph and hence the performance was appreciable in those cases. For instance, Parallelized Multilevel Bidirectional Arcflags achieved a speedup (with respect to run time) of nearly 2.91 on real world data sets while its performance was considerably low on the planar graphs (1.33) generated by the same library, LEDA.

TABLE II
COMPARISON OF SPEEDUP WITH RESPECT TO VERTEX VISIT COUNT

| | Speedup for planar graph | Speedup for random graph | Speedup for real world data set |
|---|---|---|---|
| Arcflags | 1.61 | 1.023585 | 1 |
| Bidirectional | 0.69 | 1.22 | 1.2368 |
| Bidirectional Arcflags | 2.5571 | 1.33 | 1.8125 |
| Parallel Arcflags | 1.50 | 1.02 | 1.3437 |
| Parallel Bidirectional | 1.22 | 1.40 | 1.2973 |
| Parallel Bidirectional Arcflags | 1.13 | 1.06 | 2.3888 |
| Multilevel | 1 | 1 | 1 |
| Parallel multilevel | 1 | 1 | 1 |
| Multilevel Arcflags | 1.002317 | 1.0206 | 1.228 |
| Parallel Multilevel Arcflags | 1.4272 | 1.0213 | 3.307 |
| Bidirectional Multilevel Arcflags | 1.7888 | 2.877 | 3.066 |
| Parallel Bidirectional Multilevel Arcflags | 1.867 | 2.68 | 3.211 |

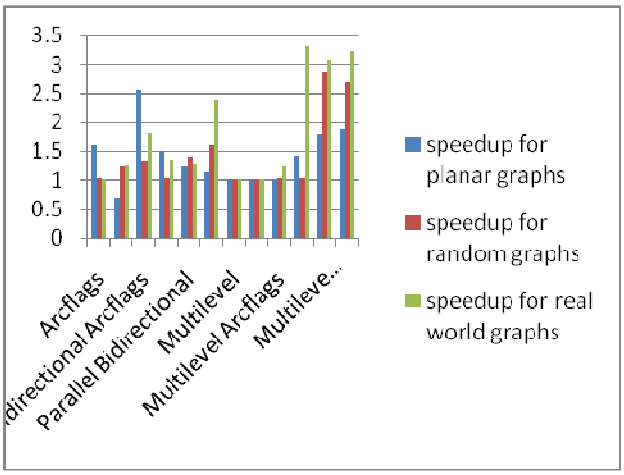


Figure 2. Comparison of speedup with respect to vertex visit count.

The performance of this technique is also appreciated with respect to number of nodes visited, wherever hop count is a Qos parameter. The proposed speed up technique (Parallelized Multilevel Bidirectional Arcflags) was able to perform better under the same experimental setup compared to the other techniques. The performance was also seen to have improved on real world graphs compared to the graphs generated by LEDA.

The performance of various combinations of speedup techniques with random and planar graph are shown in Fig. 3 to 8. The pre-processing time is used as the performance parameter in all the graphs.

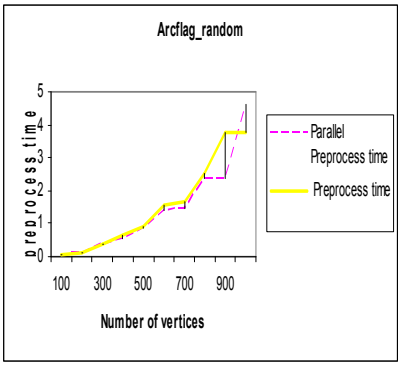


Figure 3. Arcflag random graph.

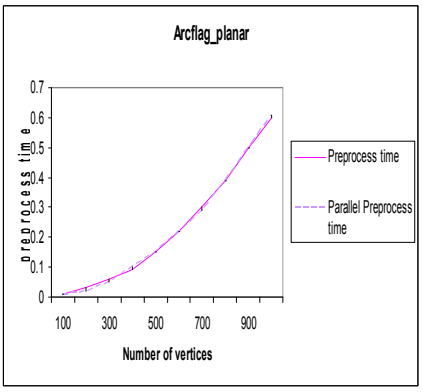


Figure 4. Arcflag Planar Graph.

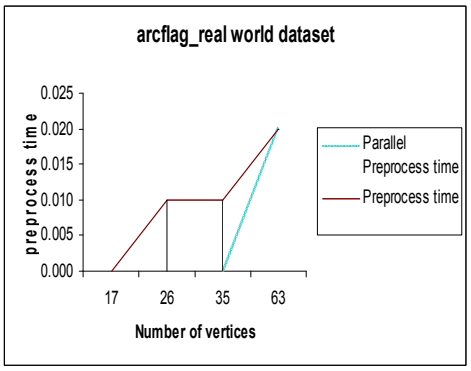


Figure 5. Arcflag Real World Dataset.

The code segments which were parallelized in the pre-processing phase have reduced the pre-processing time to a considerable amount. It is comparatively good in real world graphs than other types of graphs.

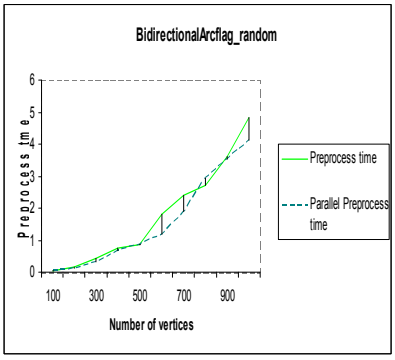


Figure 6. Bidirectional Arcflag random graph.

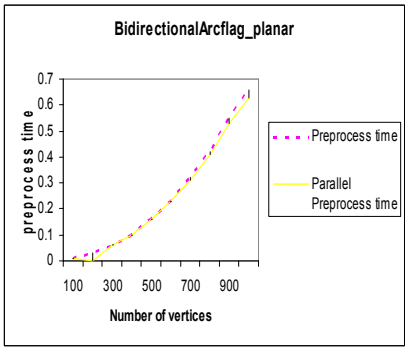


Figure 7. Bidirectional Arcflag planar graph.

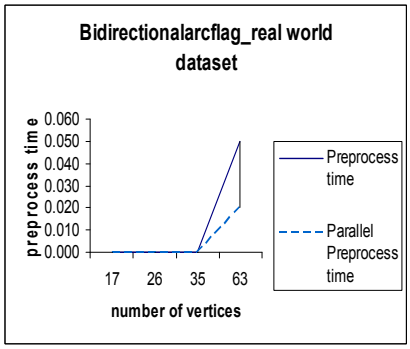


Figure 8. Bidirectional Arcflag real world datasets

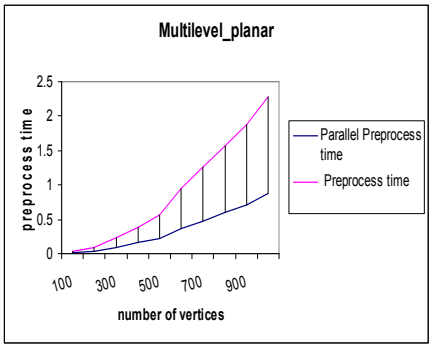


Figure 9. Multilevel Planar graph.

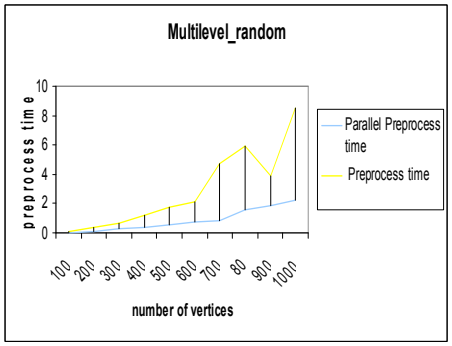


Figure 10. Multilevel Random graph.

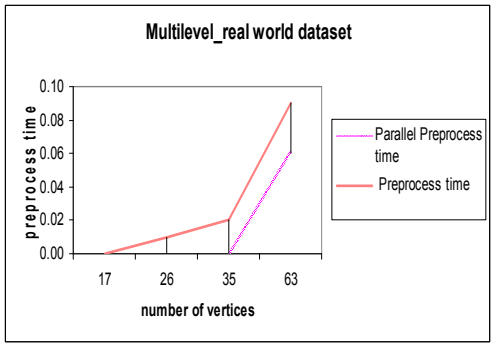


Figure 11. Multilevel Real World Dataset graph.

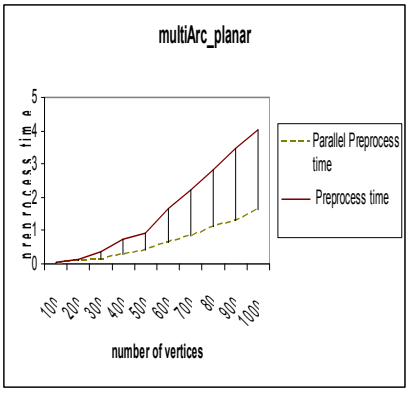


Figure 12. Multilevel Arcflags Planar graph.

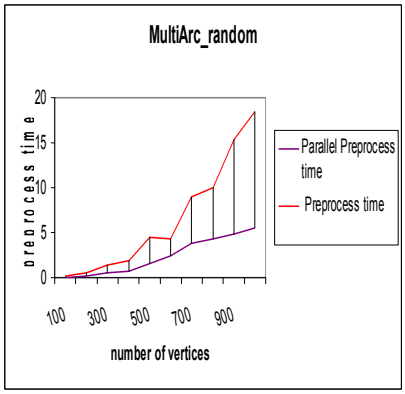


Figure 13. Multilevel Arcflags Random graph.

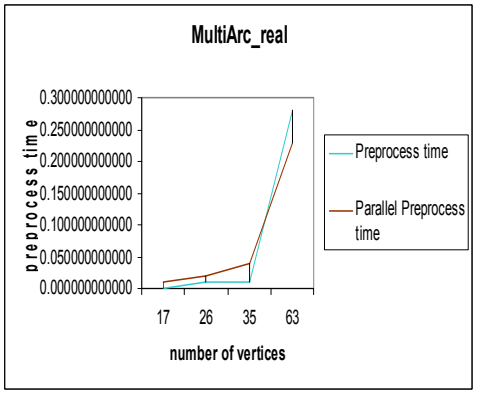


Figure 14. Multilevel Arcflags Real World Dataset

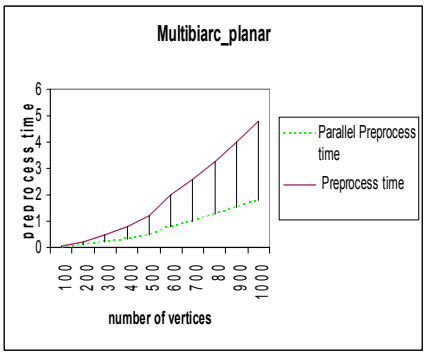


Figure 15. Multilevel Bidirectional Arcflags Planar graph.

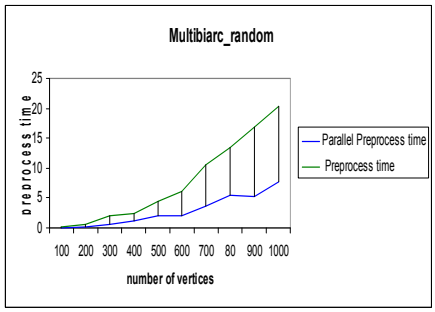


Figure 16. Multilevel Bidirectional Arcflags Random graph.

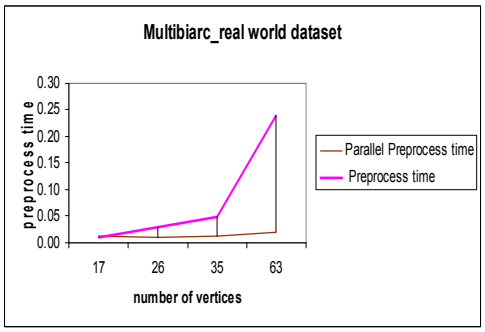


Figure 17. Multilevel Bidirectional Arcflags Real World Dataset

The parallelization for the preprocessing phase is done using multilevel technique, arc flag technique and its combination. The results are represented as charts and it shows that the combined technique for real world data set gives better results on a relative basis. Even though the time for pre-processing is high in some of the techniques, the time is effectively saved in the shortest path computation phase in those cases because of parallelism.

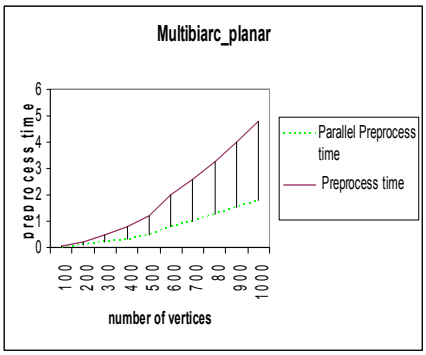


Figure 15. Multilevel Bidirectional Arcflags Planar graph.

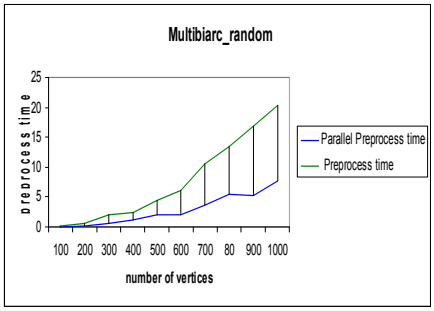


Figure 16. Multilevel Bidirectional Arcflags Random graph.

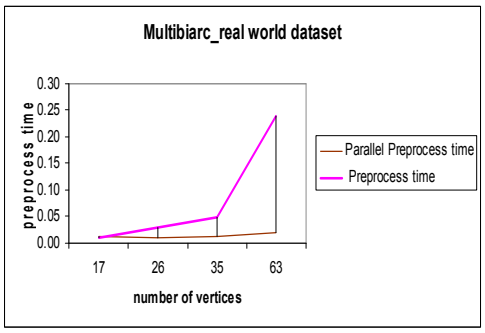


Figure 17. Multilevel Bidirectional Arcflags Real World Dataset

The technique of combining Bidirectional arc flags with multilevel approach(COBAM) achieves a very good speedup(≈3) in road networks, moderate speedup(1 and above) in random graphs and poor speedup(<1) in planar graphs with respect to runtime. With respect to number of vertices visited it(COBAM) achieves a speedup of equivalently better i.e, a very good speedup(3 and above) in road networks, moderate speedup((≈3) in random graphs and poor speedup((≈2) in planar graphs

.

## 5. CONCLUSION

The optimization technique works well for combining three speedup techniques namely bidirectional search, Multilevel approach and Arc flag method. The new speedup technique performs well on all three types of graphs namely random, planar and real world graphs. The performance of the new speedup technique is extremely good on real world graphs. Preprocessing phase considerably improved the speedup of the system by reducing the runtime of the technique and reducing the number of nodes visited during the shortest path computation.

The optimization can be extended with other types of real world graphs and new combinations. Various partitioning strategies can also be considered in arc flags to improve the performance of the combining speedup technique.

## References

[1] DIJKSTRA, E. W. (1959) 'A note on two problems in connection with Graphs', In Numerische Mathematik, Vol. 1, Mathematisch Centrum, Amsterdam, The Netherlands, pp.269–271.

[2] Frank Schulz, Dorothea Wagner, and Weihe, K. (2000) 'Dijkstra's algorithm on-line: An empirical case study from public railroad transport', ACM Journal of Experimental Algorithmics, Vol. 5.

[3] Holzer, M, Schulz. F, Wagner and Willhalm. T. (2006) 'Combining speed-up techniques for shortest-path computations', ACM Journal of Experimental Algorithmics, Vol.10, Article No.2.5, pp.1-18.

[4] Dorothea Wagner and Thomas Willhalm. (2007) 'Speed-Up Techniques for Shortest-Path Computations', In Proc. STACS 2007, LNCS , Springer-Verlag, New York. pp43- 59.

[5] GUTMAN, R.J. (2004) 'Reach-based routing: A new approach to shortest path algorithms optimized for road networks', In Proceedings of the Sixth Workshop on Algorithm Engineering and Experiments and the First Workshop on Analytic Algorithmics and Combinatorics.

[6] Frank Schulz, Dorothea Wagner, & Christos Zaroliagis. (2002) 'Using multi-level graphs for timetable information in railway systems', In Proc. 4th Workshop on Algorithm Engineering and Experiments. LNCS 2409, Springer-Verlag, New York. pp43- 59.

[7] I. Phol. (1971) 'Bi-directional Search', In Machine Intelligence,volume 6, pp 124-140. Edinburgh Univ. Press, Edinburgh

[8] Andrew V.Goldberg and Chris Harrelson. (2005) 'Computing the Shortest Path: A* Search Meets Graph Theory', In Proc. 16th Annual ACM-SIAM Symposium on Discrete Algorithms.

[9] Andrew V. Goldberg and Renato F. Wernecky. (2005) 'Computing Point-to-Point Shortest Paths from External Memory', In Proc. Of The Seventh Workshop on Algorithm Engineering and Experiments
(ALENEX05).

[10] Sanders, P. and Schultes. D. (2005) 'Highway hierarchies hasten exact shortest path queries', In the Proceedings European Symposium on Algorithms.

[11] Sanders, P. and Schultes, D. (2006) 'Engineering highway hierarchies', In the Proceedings of the 14th European Symposium on Algorithms. LNCS,vol. 4168. Springer, New York. Pp.804–816.

[12] Schultes. D and Sanders. P. (2007) 'Dynamic highway-node routing', In
Proceedings of the 6thWorkshop on Experimental and Efficient algorithms,LNCS. Springer, New York pp.66–79.

[13] Mohring, R. H., Schilling, H., Schutz, B.,Wagner. D., and Willhalm, T. (2006) 'Partitioning graphs to speed up Dijkstra's algorithm', ACM Journal of Experimental Algorithmics, Vol.11, Article No.2.8, pp.1-29.

[14] Dorothea Wagner and Thomas Willhalm. (2005)'Geometric Containers for Efficient Shortest-Path Computation', ACM Journal of Experimental Algorithmics, Vol.10, Article No.1.3, pp.1-30.

[15] Martin Holzer, Frank Schulz and Dorothea Wagner. (2008) 'Engineering Multilevel Overlay Graphs for Shortest-Path Queries', ACM Journal of Experimental Algorithmics, Vol.13, Article No.2.5, September.

[16] Bauer. R, Delling. D, Sanders. P, Schieferdecker. D, Schultes. D &Wagner. D. (2008) 'Combining hierarchical and goal-directed speed-up techniques for dijkstra's algorithm', in the proceedings of the 7th
Workshop on Experimental Algorithms(WEA'08), Springer, Berlin,pp.303-318.

[17] Reinhard Bauer, Daniel Delling, Peter Sanders, Dennis Schieferdecker, Dominik Schultes, & Dorothea Wagner. (2010) 'Combining hierarchical and goal-directed speed-up techniques for Dijkstra's algorithm', ACM Journal of Experimental Algorithmics, Vol. 15, Article No. 3.

[18]Dominik Schultes, Johannes Singler, Peter Sanders. (2008)'Parallel Highway Node Routing', A Technical Report, Feburuary. algo2.iti.kit.edu/schultes/hwy/parallelHNR.pdf

[19]R.Kalpana, P.Thambidurai, Arvind Kumar, R. Parthasarathy, and Praful Ravi. (2010), 'Exploiting Parallelism in Bidirectional Dijkstra for Shortest-Path Computation', in the Proceedings of International conference on Computers, Communication and Intelligence at , Vellammal college of Engg., & Tech.,Madurai, India, pp. 351-356, July.

[20]R.Kalpana, P.Thambidurai,(2010), 'Optimization of Landmark preprocessing with Mulitcore Systems', Journal of Computing, Vol.2, Issue.8, pp.102-108, August.

[21]R.Kalpana, P.Thambidurai (2011), 'Optimizing shortest path queries with parallelized Arc flags', in the Proceedings of IEEE International conference on Recent trends in Information Technology, MIT Campus,Anna University, Chennai, India, June.

[22] 'The OpenMP - API specification for parallel programming', available at http://www.openmp.org

[23] The Advanced Computing Systems Association (2000) 'Amdahl's law & Parallel Speedup', http://www.usenix.org/publications/library/proceedings/als00/2000papers/papers/full_papers/brownrobert/brownrobert_html/node3.html

[24] Algorithmic Solutions Software GmbH (1995) 'LEDA', available at http://www.algorithmic-solutions.com