OPTIMAL SOLUTION FOR SHORTEST PATH PROBLEM USING HEURISTIC SEARCH TECHNIQUE

Mr. Girish P Potdar, Dr. R C Thool

Abstract— **Search problems can be classified by the amount of information that is available to the search process. Heuristic search techniques use problem specific knowledge, beyond the problem definition itself. These methods make use of evaluation function, to pick up next best possible node during the process of reaching to the goal state. The same principle can be applied to various problems, including the shortest path finding. The basic philosophy of A*, a heuristic algorithm, can easily be used to find the shortest path between given pair of nodes. Existing heuristic search algorithms have been found to make use of, QUEUE implementation scheme, OPEN/CLOSE LIST implementation scheme, Ordered list implementation and Generalized link list approach.**

Use of data structure definitely affects the performance of the base algorithm. The proper data structure with heuristic algorithm will give improved performance with respect to the number nodes being explored. This will in turn result in improved time and space complexities.

In this paper, we present case study of finding the shortest path using heuristic search technique with multilevel link list as data structure. The scope is confined to the analysis of the case study and discuss on the optimality of the method.

The experiment was carried out for finding the shortest path between a pair of cities represented as graph. The results indicate that the total search space gets reduced in each iteration and also the shortest path is always gives an optimal solution and terminates with success.

Index Terms—**OPEN/CLOSE LIST, Multilevel linked list, Optimization, heuristic cost function, A* algorithm.**

I. Introduction

Heuristics is the study of the methods with rules of discovery and invention. In state space search, heuristics define the rules for choosing branches in a state space that are most likely to lead to an acceptable solution. The advantage of heuristic search over traditional blind dynamic programming is that it uses an admissible heuristic and intelligent search control for solving the problem for applicable states, given the start and goal states. While doing this it ignores the non-reachable or irrelevant parts of the state space [4].

There are many existing heuristic search algorithms like hill climbing, A*, AO*, and simulated annealing. Hill climbing suffers from problems like local maximum, plateau, and ridge. The graceful decay of admissibility is the major drawback of A* algorithm. AO* has overhead of expanding the partial graph one state at a time and re-computing the best policy over the graph after each step. A* algorithm is often used for finding optimized solutions, which works on the principle of “Best – First”. The effectiveness of A* algorithm, however depends heavily on the cost function. A* is optimal over admissible unidirectional heuristic search algorithms using some information under certain conditions. It implies that it will never expand more nodes than other unidirectional search algorithms [2][5][6].

When we are working in either a state space or a problem-reduction representation, achieving the required goal can be viewed as finding an appropriate finite sequence of applications of the operators. Most of the search algorithms aim only in finding goal state, though one can even determine sequence of operations leading to goal state. In conventional approach, algorithms use either Queue implementation technique or OPEN/CLOSE list technique for finding solutions. These techniques make redundant access to the linked lists, which increase the overhead. To reduce this complexity we propose a method that use a data structure which aims at making a good use of memory space without compromising much on the completeness and optimality of the algorithm [1].

The use of heuristics is very common in real world implementations. For many practical problems, a heuristic algorithm may be the only way to get good solutions in a reasonable amount of time [7].

II. PROBLEM OF SHORTEST PATH FINDING

The research work done in past have come up with different solutions for solving shortest path problem using heuristic search technique. We can visualize this problem of shortest path finding as a complete graph G (V, E). Here V denotes set of nodes representing the cities, E the set of edges denotes the path between cities and the associated cost/time/distance is given by a matrix V x V. We wish to optimize the cost/time/distance of final path [10][16].

We can, therefore define function for any v_i v_j, v_k ∈ V, with the assumptions:

$C(v_i, v_j) \geq 0 \ldots\ldots\ldots\ldots\ldots\ldots (i)$

$C(v_i, v_j) = C(v_j, v_i) \ldots\ldots\ldots\ldots\ldots (ii)$

$C(v_i, v_j) = C(v_i, v_k) + C(v_k, v_j) \ldots\ldots (iii)$

Here, $C(v_i, v_j)$ represents the cost of edge (v_i, v_j).

Thus, for any E' ⊆E, we define the length E' to be
Z (E') $=\sum_{(v_i,v_j)\subseteq E'} C(v_i, v_j)$ …. (iv)

Assumption (i) implies that there may or may not exist a direct edge (path) between nodes (cities) v_i and v_j. Assumption (ii) indicates that it is an undirected graph. The last assumption ensures that the selected edge v_i, v_j is having the least cost [3].

Vi be starting node (Start state) and Ve be goal node (goal state), where Vi, Ve ∊ V, our aim is to minimize the value of Z given equation (iv).

Now the transition from one state to another state depends on the set of applicable actions; such as minimum distance, minimum cost, minimum time etc. depending on the need. Let A denote all the applicable actions. Thus for a node v ∊ V, which is not a goal node, a (v) ⊆ A represents the applicable action at all nodes v. The objective of the problem is to determine the sequence of actions, when applied to the starting node results in the goal node. One may associate cost parameter C (v, a) to each applicable actions so that the problem now reduces to minimizing function: minimize$\sum_{i=0}^{n} C(vi, ai)$ [3].

The traditional approaches for solving shortest path problem, take the best available move at a given instance. This however may not lead to the required goal state all time. If, we get some kind of evaluation function which will serve as future informatics, then we may overcome this problem. This basic idea of using evaluation function resulted in few heuristic search algorithms such as best –first search, A* an extension of best-first search etc. These Algorithms make use of heuristic function to decide next possible choice [13].

III. FINDING SHORTEST PATH USING A* - ALGORITHM

The problem of finding shortest path between pair of nodes has various applications in the field of engineering and science. For the past years researchers have not only developed the effective solutions but also have given efficient algorithms to solve these problems. The developments in the field of artificial intelligence, has drawn attention of many researchers in modelling problems involving searches [13,18].

The shortest path problem, as we know, tries to search the best next node in the process of reaching to the goal node. Since it involves searching and in large problem domain, one might make use of heuristic searching techniques for the same A* heuristic search technique can be used to solve this problem effectively. In this case it makes use of simple calculation of estimated cost value f as a function of g and h; where f=g+h. Where g(n), represent cost of choosing the path from starting node to node n; and h(n) represents optimal cost of node n to the goal node. However the value of h(n) will be unknown in most of the situations, which results in unknown value of f(n). A* algorithm, however makes a best approximation for h(n) [9][12][15].

The traditional methods to solve this problem work on the entire state space, but the A* heuristic search algorithm tries to find solution by pruning large parts of the remaining state space [7]. The conventional method of A* algorithm to solve the shortest path problem has been given in [17][11]. The flowchart in figure 1, explains the steps involved in finding the shortest path using A* method.

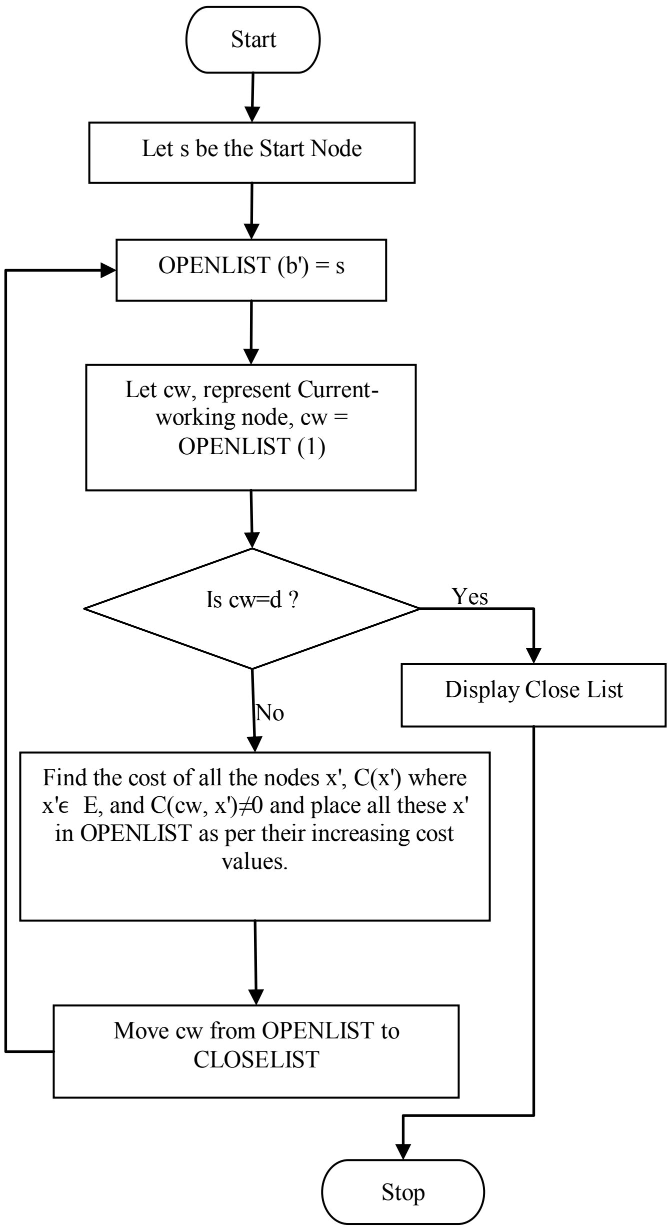
Start
Let s be the Start Node
OPENLIST (b') = s
Let cw, represent Current-working node, cw = OPENLIST (1)
Is cw=d ?
Yes
No
Display Close List
Find the cost of all the nodes x', C(x') where x'∊ E, and C(cw, x')≠0 and place all these x' in OPENLIST as per their increasing cost values.
Move cw from OPENLIST to CLOSELIST
Stop

Fig. 1 Flow of A* Method.

Let us see the working of this algorithm with the help of an example. Consider the graph as shown in figure 2, we wish to find shortest path between A and F [11].

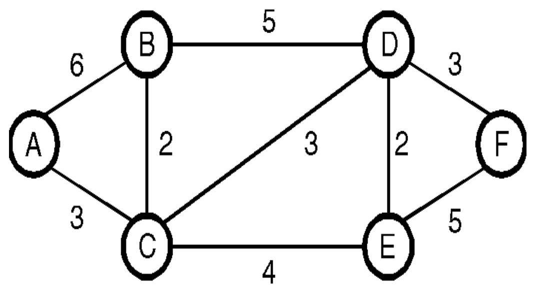

Fig. 2 Graph to illustrate function of method discuss.

The computations involved in finding shortest path from node A to node F are as given in figure3 to figure 5.
Start with the source node or start state: A
Exploring this node we have:

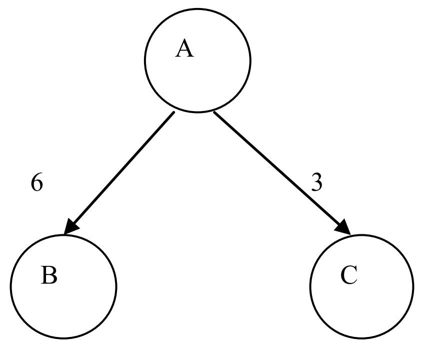

Fig. 3: First level of solution tree.

g(B) = 6 ;h(B)= min {2,5} = 2;f(B)= 6+2 = 8
g(c) = 3;h(C)= min{3,4} =3; f(C)= 3+3 = 6;
Since f(C) is minimum, use C for next level expansion

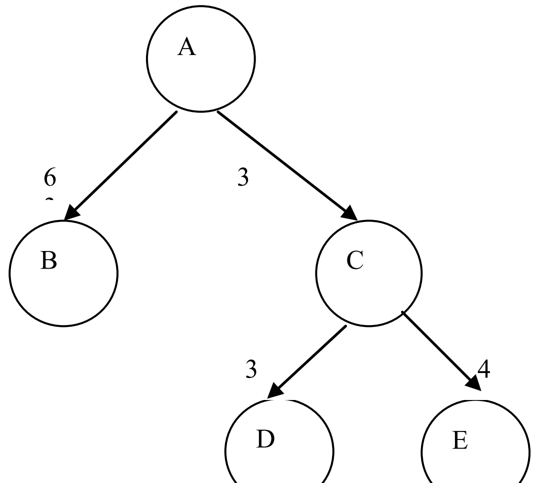

Fig. 4: Solution tree of the expanding C.

g(D) = 3+3=6 ; h(D)= min {2,3} = 2;f(D)= 6+2 = 8
g(E) = 3 + 4 =7; h(E)= min{5} = 5;f(E)= 7+5 =12;

As f (D) is minimum expand D

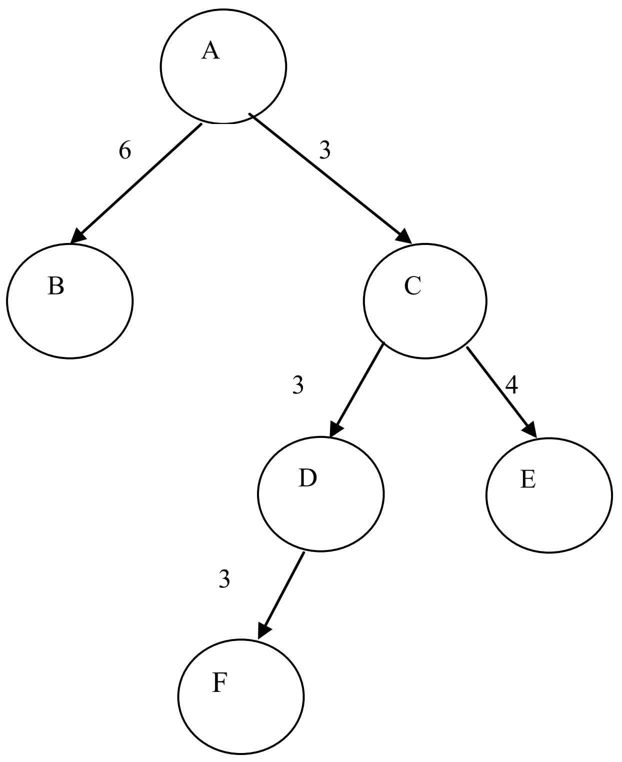

Fig. 5: Solution tree of the expanding D.

g(F) = 6+3 = 9 ; h(F)= F is the goal state, stop.

The updates in OPEN and CLOSE lists when we apply the above mentioned A* algorithm on the graph given in figure 1. We get:

OPEN : empty
CLOSE : empty

OPEN : A
CLOSE : empty

OPEN : B, C
CLOSE : A

OPEN : D, B, E
CLOSE : A, C

OPEN : B, F, E
CLOSE : A, C, D

F= goal state; STOP.

IV. MULTI LEVEL LINKED LIST APPROACH

A. Methods

Multilevel linked list (MLL) is used in the proposed algorithm. MLL maintains a parent list and successor list. Every parent node maintains its list of successors. As we move up in the parent list, the previous successor lists (in most of cases) become obsolete and may be freed. This will result in better memory utilization.

We now look into the method. This method assumes that graph representing the connectivity is almost fully connected, without direct path from source to goal state.

Algorithm I:

Basic Notation

s = Staring node.
d = Destination node.
S = set of nodes in parent list of MLL.
C = current node which is being expanded (which is at the top of S).
S' = set of adjacent nodes of c.

gc(x) = actual cost between c and x, where $x \in S'$.
hc(x) = min(gc(x)) $\forall x \in S'$.

The algorithm:

```
step 1:   Place s in MLL.
step 2:   C=s and C will be now added in S;
          Totalcost=gc(x).
step 3:   Find S' of C.
step 4:   ∀x' ∈ S' Compute f(x'), where f(x')=gc(x') + hc(x').
          Now choose the least f(x) from available values of
          f(x').
          C = x' and C will be now added to S.
step 5:   Clear S'.
step 6:   Totalcost = toalcost+gc(x').
          Repeat Step 3 to Step 6 till C≠d.
          The final value of Totalcost will be the shortest
          path cost.
```

Considering the same example given in figure1, the entire process of creation of MLL is depicted in figure6 to figure7.

First Node A, start node is chosen.

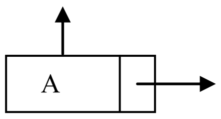

Fig. 6: Initial MLL representation.

Now adjacent nodes of B, C are added to MLL

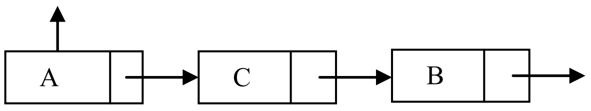

Fig. 7: MLL after adding successors of A.

As C is next best feasible node. It is taken for expansion.

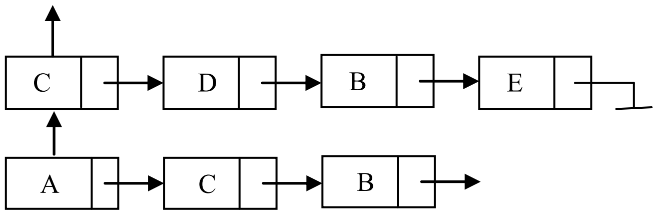

Fig. 8: MLL after including C in parent list.

Again the heuristic value indicate 'D' as best choice.

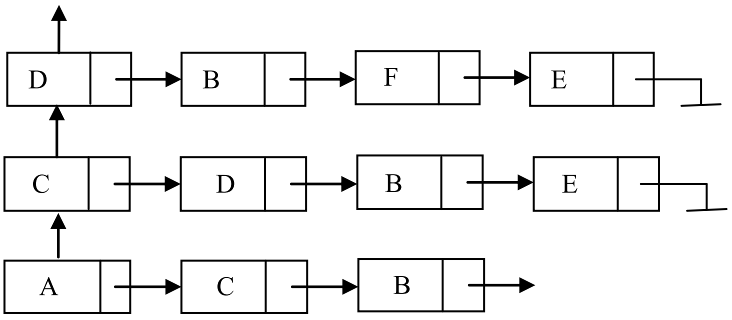

Fig. 9: MLL after including D in parent list.

Now, as we encountered F, which is the required goal node, the process terminates successfully. The final path will be A→C→D→F.

B. *Optimality of the Method*

The optimality of MLL, the method presented in section 4, is heavily bounded by the optimality of A* itself. We have made use of the heuristic cost function f=g+h. Many efforts have been already done in this regard [6,7,13]. Few authors have claimed that the A* algorithm is a special kind of branch and bound strategy. A* algorithm selects the node k, for expansion only after it finds that f(k)<=c, where c is lowest cost functions available at the instance when search reaches to node k. This implies that A* will ensure optimal path [11][19]. On the contrary, if there exists a situation, where in upon reaching node k', somewhere from start node to goal node, the cost function at k' might be more than that if another node would have been chosen for expansion. It is thus apparently clear that the computation of f did not have the adequate information in such cases. But as we are making use of absolute cost values in g and h values such a situation may not occur. It can be easily seen that the algorithm terminates when all nodes which are in parent list have bounding value f. The nodes which are in the parent list are bounded by the feasible solution. It implies that the other nodes which were in successor list must be having their f values greater than the ones in parent list and hence they are lower bounds. The results indicate that MLL approach always yields in an optimal solution. At any instance of processing, the current upper bound will be smaller or equal to the rest of the lower bounds. Let us see this argument with a bit of elaboration.

Assume that the graph with n-nodes represents the n cities and we are interested in finding shortest path from start node s to a goal node d, using the A* with MLL. The figure 10 shows the typical expansion phase.

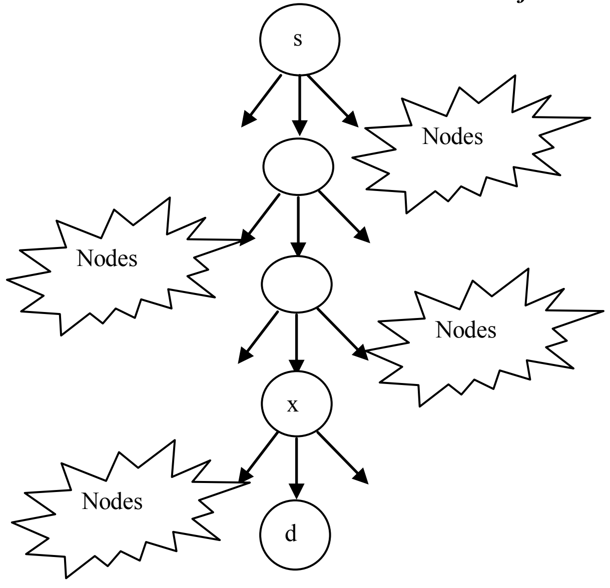

Fig. 10: A* algorithm with MLL -expansion process.

Assuming that, the nodes are expanded with optimal cost starting from s, and we have reached upto node x. Now if there is a path going from x to d, then

h(x) = g(d)-g(x)
h(x) = h'(x),
Since h(x) is least one leading to goal node d.

$$\therefore f(d) = g(d) + h(d) = g(x) + h'(x) = f'(x)$$
$$g(d) = f(d) = f'(x).$$

Here we have already chosen x for expansion, hence f'(x)<=f(x') iff x' is an expanded node, As per previous discussion we know f(x')<=f(x), hence g(d)=f'(x)<=f'(x') and thus must be an optimal solution.

Another interesting question that can be raised here is: "Does the MLL based A* algorithm always terminates? If yes, does it ensures the optimal solution?" Theorem – I, addresses this case. This theorem is based on the identical lines of proof given in [6].

C. Theorem-I: Any time MLL based A algorithm terminates by finding the optimal solution to shortest path.*

Proof: This can be proved in two parts: Algorithm always terminates and that the algorithm cannot terminate without finding the optimal solution. Let us first prove that the algorithm cannot terminate without finding the optimal solution. Let us assume that algorithm terminates before finding the optimal path with path cost of f'(x), where x may or may not be a goal node.

Let C0, C1, C2, …. Ck be the heuristic cost value of reaching to node i from the previously traversed path i.e. C1 is path cost incurred from reaching to C1 from C0 and so on. (Note here that path cost of reaching to Cj from Cj-1 doesn't imply direct edge-cost of CJ-1 Cj, but it is lowest path cost to reach Cj from the previously traversed lowest path cost).

We know C0<C1<C2……Ck<f'(x); the inequality Ck<f'(x) is true if the algorithm terminates before reaching to an optimal solution. Now consider the path P' from start node to a goal node. With the assumption that this optimal solution path was not found, implies that there must have been some node n along this path that was generated but the algorithm did not expand. This is possible if g(n)+h(n)<=bk.

Since, we are accounting for all possible neighbor nodes and then choose the node for expansion not only based on g values but also on the corresponding ha values, Thus we know that g(n)+h(n)>=f'

∴k; g(n)+h(n)>= f'>bi.

From this we can claim that the algorithm will not terminate before an optimal solution is found. Another way of proving the same is presented in Theorem – II.

D. Theorem – II: A with MLL yields optimal path.*

Proof: To prove this we need to consider two cases.
Case I: Inherently A* itself is optimal.
Case II: Use of MLL does not affect Case –I.

Proof for case I: A* algorithm makes use of heuristic cost function. The method discussed in section 4 also uses heuristic evaluation function f(n)=g(n)+h(n) for finding the shortest path; where g(n) is the edge cost between node n and its adjacent node; h(n) is the minimum cost to next reachable node from n. While choosing next node for expansion minimum of h(n) is accounted in determining f(n). The edge cost being an absolute value, the value of h(n) is consistent. It also results in non-decreasing values of f(n) along any path. Thus value of f(n) plays significant role in selecting the best next node for expansion at all levels, till we reach the goal node.

Now, the only question to be addressed is to assess the correctness of the node n being chosen falls on optimal path (already we have shown this in theorem-I), follows the method of contradiction: A node n has been chosen for further expansion but the optimal path to n itself has not been identified. It indicates that the optimal path to node n, must have unexpanded nodes (which again is contradiction that there cannot be shorter path going only through expanded nodes so far). Hence there must have been node x, which might have been ignored during the process of expansion. But the value of f along the possible paths (due to the values of h) is non- decreasing, x would have been chosen ahead of the node n which again is a contradiction. Hence, the series of nodes chosen as a first goal node must have optimal cost path length, since the f(n) is the real path cost for n=goal node (h(goal node)=0). The other paths that are having either same path cost of the current optimal path cost of greater. Hence it proves that the algorithm always results in the optimal path.

Proof for case II: This proof now tends to mean that the use of MLL does not affect the optimality exhibited by the base method. We can prove that the use of MLL does not contradict in choosing the next node for expansion.

In MLL, the successor nodes of current node n are added in its successor list, according to their respective f values. While computing f, the similar strategy as in case-I is adopted. The next node for expansion is selected as the first node x from the successor list of the current node in the parent list. When we reach down to a goal node with a path cost of P, it has to be the optimal cost. If this is not true, then the choice of x, at some level, for further expansion is wrong. It implies that the method of creating or adding nodes in successor list is incorrect. However the list being created is purely based on f value, there cannot exist another node x' whose f' value is smaller than f and got placed later in the list. Thus it is clear that MLL does not conflict the optimality value computation.

Hence from case-I and case-II it is clear that use of MLL with A* method always results in optimal solution.

The significant change that we observe is the use of MLL reduces nodes/edges being visited or explored.

V. Case Study Results

The MLL based heuristic search technique was implemented and tested of Intel C2D system (2.44 GHz, 2GB RAM). Extensive results are given by the same authors in [17]. The dataset links used here for testing consists only of all existing national highways between given two cities (Maharashtra state). For example between Nagpur and Mumbai, we have the possible road connectivity as given in figure 11, 12 and table 1 (actual data collected from Google maps).

The corresponding distance matrix for above example is presented in Table 1

Distance (KM)	Nagpur	Nanded	Karanja	Khamgaon	Dhule	Nashik	A Nagar	Mumbai
Nagpur	0	335	221	156	521	0	0	0
Nanded	0	0	0	0	0	0	315	0
Karanja	0	0	0	0	0	0	435	0
Khamgaon	0	0	0	0	0	0	333	0
Dhule	0	0	0	0	0	158	0	0
Nashik	0	0	0	0	0	0	0	168
A Nagar	0	0	0	0	0	0	0	239
Mumbai	0	0	0	0	0	0	0	0

Table 1 Example 1 Road Distance Matrix.

The graphical results showing the shortest path from Nagpur to Mumbai is:
Star Node: Nagpur
End Node: Mumbai

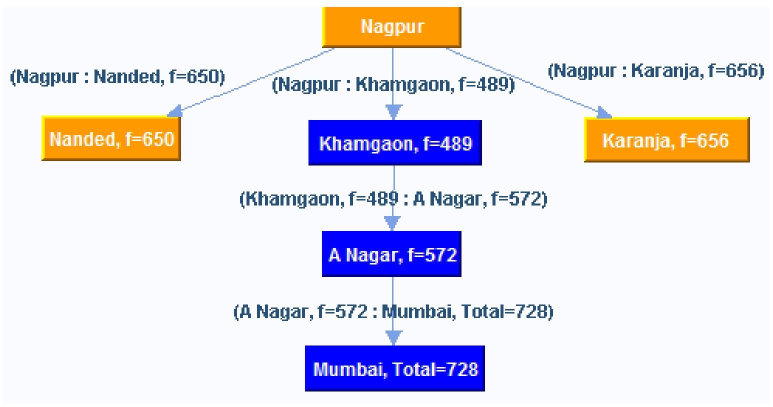
Nagpur
(Nagpur : Nanded, f=650)
(Nagpur : Khamgaon, f=489)
(Nagpur : Karanja, f=656)
Nanded, f=650
Khamgaon, f=489
Karanja, f=656
(Khamgaon, f=489 : A Nagar, f=572)
A Nagar, f=572
(A Nagar, f=572 : Mumbai, Total=728)
Mumbai, Total=728

Fig. 11 A* algorithm with MLL for Example 1 Graphical results.

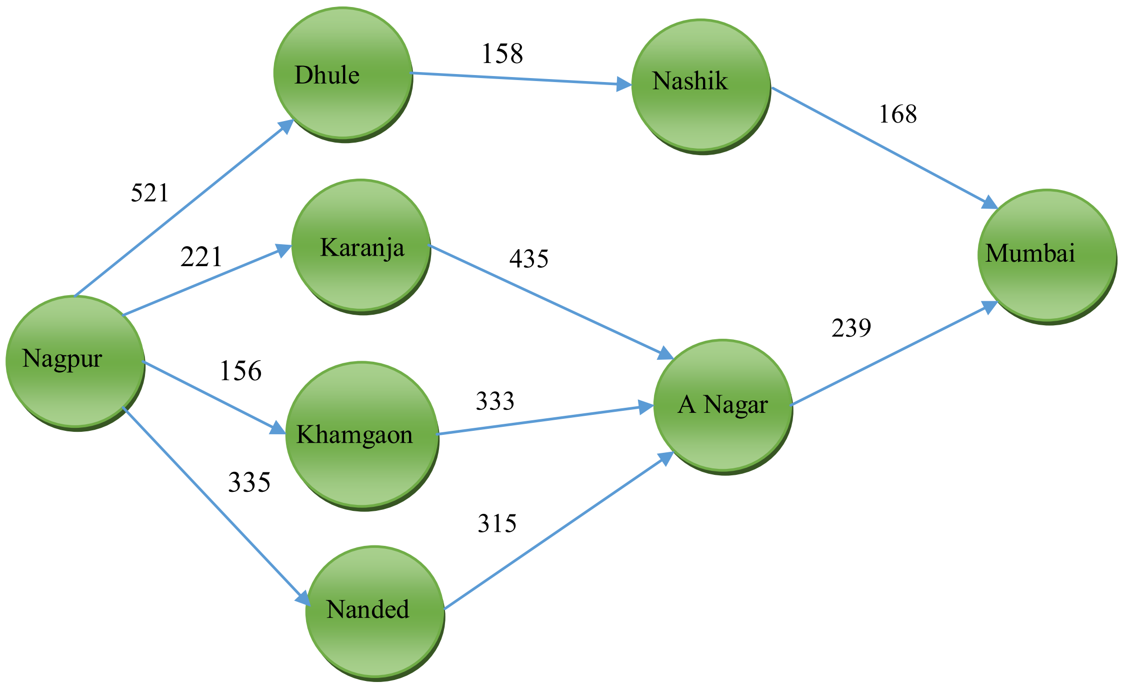

Fig. 12 Graph for Example 1

Another example involving rural road system.

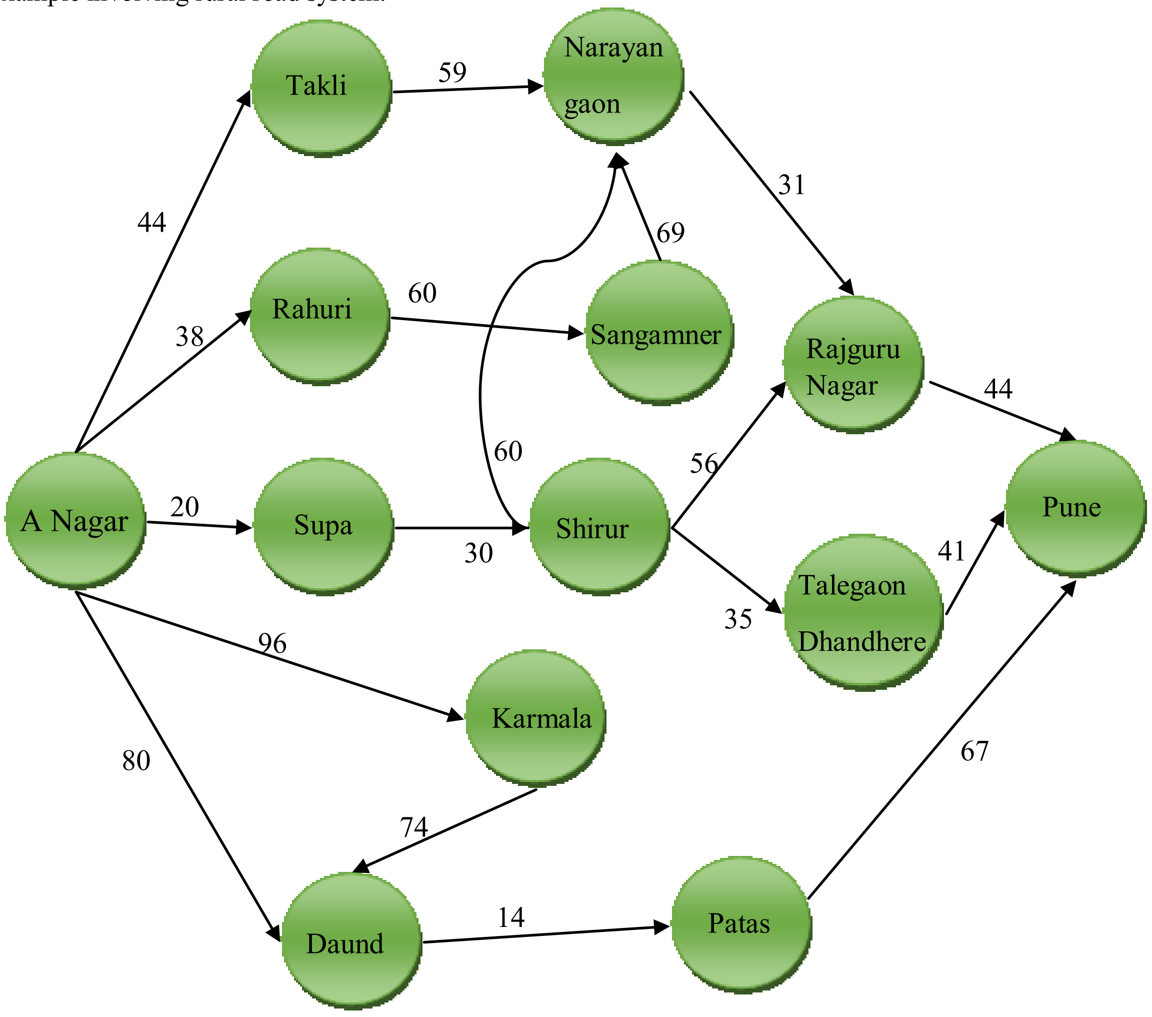

Fig. 13 Example 2- Road Graph.

Distance (KM)	A	B	C	D	E	F	G	H	I	J	K	L	M
A	0	20	44	80	38	96	0	0	0	0	0	0	0
B	0	0	0	0	0	0	33	0	0	0	0	0	0
C	0	0	0	0	0	0	0	59	0	0	0	0	0
D	0	0	0	0	0	0	0	0	14	0	0	0	0
E	0	0	0	0	0	0	0	0	0	60	0	0	0
F	0	0	0	74	0	0	0	0	0	0	0	0	0
G	0	0	0	0	0	0	0	60	0	0	35	56	0
H	0	0	0	0	0	0	0	0	0	0	0	31	0
I	0	0	0	0	0	0	0	0	0	0	0	0	67
J	0	0	0	0	0	0	0	69	0	0	0	0	0
K	0	0	0	0	0	0	0	0	0	0	0	0	41
L	0	0	0	0	0	0	0	0	0	0	0	0	44
M	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 2 Road Distance Matrix for example 2.
(Note: A=Nagar, B=Supa, C=Takli, D=Daund, E=Rahuri, F=Karmala, G=Shirur, H=Narayangaon, I=Patas, J=Sangamner, K=Talegaon Dhandhere, L=Rajgurunagar, M=Pune)

Start Node: A Nagar
End Node: Pune

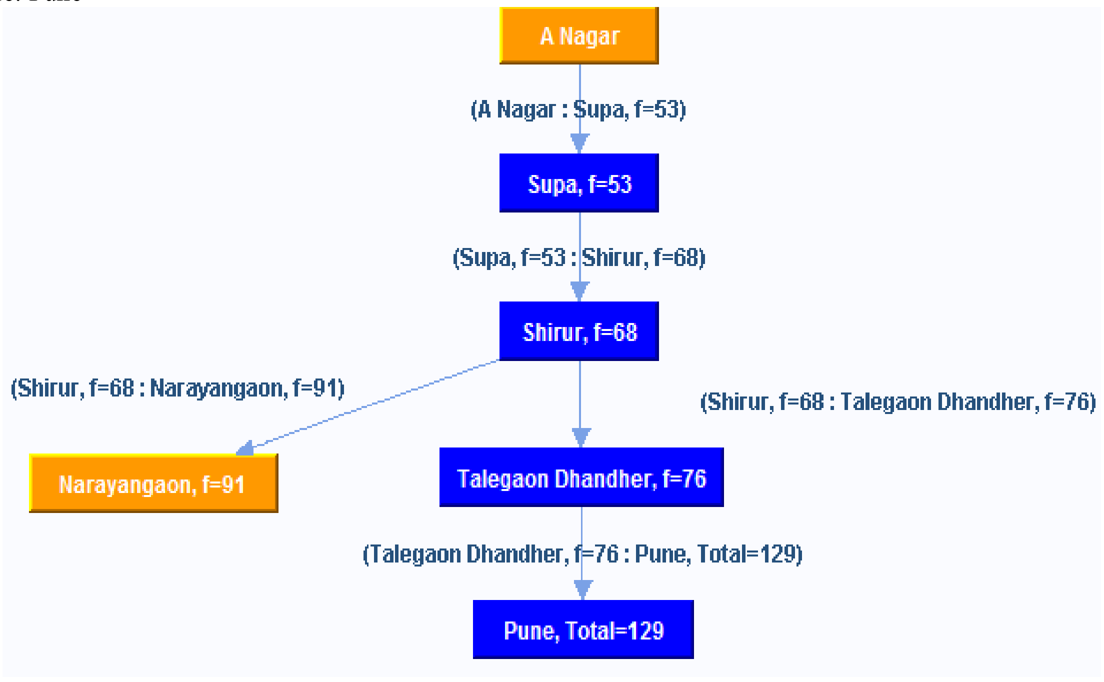
A Nagar
(A Nagar : Supa, f=53)
Supa, f=53
(Supa, f=53 : Shirur, f=68)
Shirur, f=68
(Shirur, f=68 : Narayangaon, f=91)
(Shirur, f=68 : Talegaon Dhandher, f=76)
Narayangaon, f=91
Talegaon Dhandher, f=76
(Talegaon Dhandher, f=76 : Pune, Total=129)
Pune, Total=129

Fig. 14 Graphical Results of A* algorithm Example 2 .

Star Node: Supa
End Node: Pune

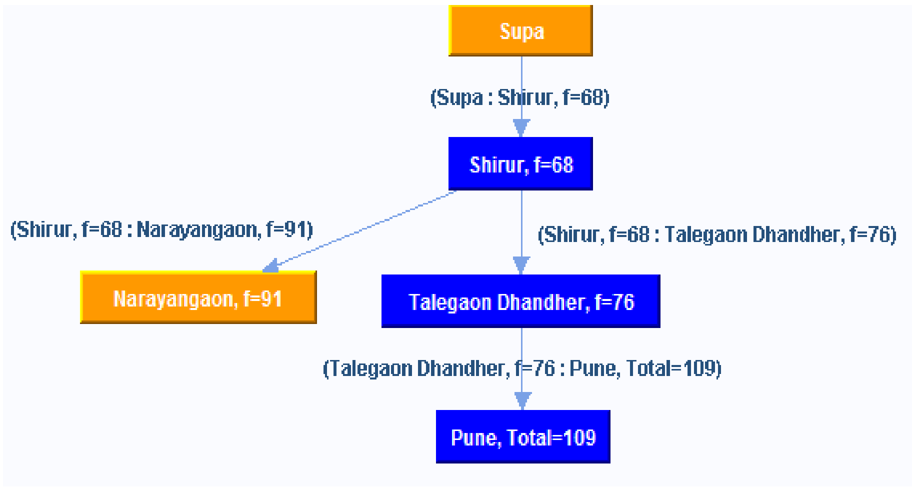
Supa
(Supa : Shirur, f=68)
Shirur, f=68
(Shirur, f=68 : Narayangaon, f=91)
(Shirur, f=68 : Talegaon Dhandher, f=76)
Narayangaon, f=91
Talegaon Dhandher, f=76
(Talegaon Dhandher, f=76 : Pune, Total=109)
Pune, Total=109

Fig. 15 Graphical result of A* with MLL algorithm for Example 2 [changed source and destination]

We did try with real data with varying number of cities, and the results indicates that the method always yielded an optimal path. Table 3 summarizes the results.

Here number of nodes indicate different set of cities. Hence we may observe in few cases, less number of nodes get explored. However, if we apply our method for increasing number of nodes (i.e. adding additional cities and links to the existing set) then we find the proportionate changes in the nodes getting explored.

For larger number of nodes, the method yields good results, as shown in [17].

No of nodes	No of nodes Explored	No of Edges Explored	No of nodes in final path#
10	8	15	5
15	21	41	8
20	29	56	8
25	24	24	2
30	64	127	10
35	61	121	10
40	52	102	8
45	87	173	8
50	121	238	9
55	77	152	7
# These numbers include start node and goal node.			

Table 3 Summary of the results on real data set.

.

VI. Conclusion

In this paper, we have presented a method using A* heuristic search algorithm to find the optimal shortest path and validated the same using real time data. The work was focused on determining the optimality of solution being obtained by our approach.

For random set of data for varying number of nodes, already we have proved that the number edges and number of nodes being expanded are less compared to other approaches [17].

In this paper we have given the results specific to a real data set and observed that the method presented gives an optimal solution for all the cases, except when there is a direct edge between source and goal node. This is the typical case of overestimating h value. In this case one may try out with other option such as not to terminate the process until, there are no explored nodes remaining whose f value is less than that of the current f value. This will definitely increase the number of nodes and edge counts. However, it may not be a realistic approach for large value of n where direct edge between start and goal node is recommended.

References

[1] Blai Bonet and Eric A. Hansen, (2010) "Heuristic Search for Planning under Uncertainty", Chapter in Heuristics, Probability and Causality: A Tribute to Judea Pearl College Publications, pp. 3-22.

[2] Eric A. Hansen, Rong Zhou, (2007) "Anytime Heuristic Search", Journal of Artificial Intelligence Research- 28, pp. 267-297.

[3] G. Cornuejols and G. L. Nemauser, (1978) "Tight bounds for christofides' traveling salesman heuristic", Short Communication Mathematical Programming, Volume 14, Issue 1, pp. 116-121.

[4] Anne L. Gardner, (Number 1, 1980) "Search: An Overview", AI magazine, Volume 2, pp. 2.

[5] R. Korf, (1990) "Real time heuristic search", Artificial Intelligence ACM Digital Library, Volume 42, pp. 189-211.

[6] Rina Dechter and Judia Pearl, (July 1985) "Generalized Best-First Search Strategies and the Optimality of A*", Journal of the Association for Computing Machinery, Volume. 32, No. 3, pp. 505-536.

[7] L. Fu, D. Sun, L. R. Rivet, (2006) "Heuristic shortest path algorithms for transportation applications: State of the art", Elsevier – Computer and Operations research- 33, pp. 3324-3343.

[8] Girish P. Potdar, Dr. R. C. Thool, (2013) "An Alternate way of implementing Heuristic, Searching Technique", International Journal of Research in Computer and Communication Technology. Volume 2, No. 9, pp. 793-795.

[9] G. Guida and M. Somalvica. "A method for computing heuristics in problem solving", Information Sciences, vol. 19, 1979, pp. 251-259.

[10] C. H. Peng, J. S. Wang and R. C. T. Lee, (1994) "Recognizing Shortest Path Trees in Linear Time", Information Processing Letters, Volume 57, pp. 77-85.

[11] R. C. T. Lee, S.S Tseng, R. C. Chang, Y. T. Tsai, (2012) "Introduction to Design and analysis of algorithms –A strategic approach", Tata McGraw-Hill

[12] P. E. Hart, N. J. Nilsson and B. Raphael, (1968) "A formal basis for the heuristic determination of minimum cost paths", IEEE Transactions on systems, Science and Cybernate, 4(2), pp. 100-107.

[13] Andrew V. Goldberg, Chris Harrelson, (2005) "Computing the shortest path: A* search meets graph theory", Proceedings of the Sixteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA-2005), pp. 156–165.

[14] R. Korf, (1993) "Linear-space best-first search", Artificial Intelligence, 1993, Volume 62, Issue 1, pp. 41–78.

[15] I. Pohl, (1970) "Heuristic search viewed as path finding in a graph", Artificial Intelligence, Volume 1, Issue 3, pp. 193–204.

[16] B. V. Cherkassy, A.V. Goldberg and T. Radzik, (1994) "Shortest path algorithms: Theory and experimental revolution", inproc 5th ACM-SIAM symposium on discrete algorithms, pp. 516-525.

[17] Girish P. Potdar and Dr. R. C. Thool, (July 2014) "Comparison of Various Heuristic Search Techniques for Finding Shortest Path", International Journal of Artificial Intelligence & Applications (IJAIA), Volume 5, No. 4, pp. 63-74.

[18] O. H. Ibarra, H. Wang and Q. Zheng, (1992) "Minimum cover and single source shortest path problem for weighted interval graphs and circular- arc graph", Proceeding of Thirtieth Annual Allerlon Conference on Communication, Control, and Computing, Universal of Illinois, Urbana, pp. 575-584.

[19] Jia-Shung Wang and R. C. T. Lee, (July 1990) "An efficient channel routing algorithm to yield optimal solution", IEEE Transaction on Computer, Volume 39, Issue 7, pp. 957-962.

Professor G. P. Potdar: Professor G. P. Potdar received his B.E. and M. E. Computer degree from Karnataka and Pune University respectively. He is currently pursuing his PhD from Swami Ramanand Teerth Marathwada University. He is HOD of Computer Department of Pune Institute of Computer Technology. He has published 13 books and 7 research Papers. He is member of IEEE, MISTE and MCSI. He has guided more than 15 P.G. students and more than 100 U.G. students. His area of interest is Data Structures, Design and Analysis of Algorithms.

Dr R. C. Thool: Dr R. C. Thool received his B.E and M.E Electronics from SGGS Institute of Engineering and Technology, Nanded (Marathwada University, Aurangabad) in 1986 and 1991 respectively. He obtained his Ph.D. in Electronics and Computer Science from SRTMU, Nanded. Presently he is working as Professor and Head of Department of Information Technology at SGGS Institute of Engineering and Technology, Nanded. His areas of research interest are Computer Vision, Image Processing and Pattern Recognition, Networking and Security. He has published more than 50 papers in National and International Journals and Conferences. Two students have completed their Ph.D. and 10+ students are working for their Ph.D. under his guidance. He is the member of IEEE, ISTE, ASABE (USA) and also a member of University Co-ordination committee of NVidia. Also, working as Expert on AICTE committee and NBA.