International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395 -0056
Volume: 03 Issue: 06 | June-2016 www.irjet.net p-ISSN: 2395-0072

A STUDY ON BEST FIRST SEARCH

[1,] M.Sinthiya, [2,] Dr.M. Chidambaram
Dept. of Computer Science
Rajah Serfoji Government Arts College

Abstract - Finding the exact search result of cloud storage is central to many operations in a variety of areas, including probabilistic reasoning and constraint satisfaction. This search space differs from those where best- first typically applied, because a solution search list is evaluated by its maximum cost instead of the sum of its users searching data costs. My thesis shows how to make best-first search admissible on max-cost best and first data for overall server data list in storage server. Also employ best-first heuristic search to reduce the memory requirement while still eliminating all irrelevant data in the search result. My thesis describes that the algorithms end the exact data list and order of magnitude. The purpose of this thesis, RFM technique is based on user search event to allocate the cost for data's in storage server, to specify loyal and ranked data list. Also it is used for classification based on minimum and maximum cost of data list to obtain useful rules for implementing effective data relationship management.

Key Words: **Best First Search , Heuristic Search , Data Mining , Tree , Shortest Path**

1.INTRODUCTION

1.1 Overview

Data structures such as trees, tries, and hash tables are used in a vast variety of applications for managing sets of distinct strings. These structures are chosen for a particular application that depends on factors such as how well they scale, memory requirements, speed, and whether it is required that the strings be kept in sort order. Text collections used in practice range from news and law archives, business reports, and collections or email, to repositories of documents garnered from the WWW. Such text collections can easily contain many millions of distinct words, and the number grows more or less linearly with collection size. The collections themselves range from gigabytes to terabytes, consisting of millions of documents or more; at the upper extreme, and the popular search engine Google indexes over a billion web pages. Practical management and querying of text collections relies on the use of efficient string data structures that offer fast insertion, search, and deletion of string keys.

Testing the structures on such large-scale data sets led to interesting discoveries. Surprisingly, on text collections from 100 Mb to 45 Gb, the relative speeds of the various structures were almost constant, Hash tables (assuming an efficient string hashing function) are the fastest of the structures, and found that performance can be significantly improved through the simple strategy of using chains to manage collisions and move- to-front within chains.

Trees are several times slower and, surprisingly, existing adaptive and balanced tree structures were no faster than standard binary trees. However, our novel variant of adaptive tree, the fab tree, yielded significant improvements in performance. Trees require inordinate volumes of memory, but are fast. The underlying principle of the burst tree is that it stores, not individual strings, but small sets of strings that share a common prefix. It might be argued that the performance on a small set of data is a major component of processing costs. One such example is, again, indexing of text collections. During the index construction process, it is necessary to identify the set of distinct words that occurs in each document, that is, it undertakes the task of per-document vocabulary accumulation. Most of the individual documents are small and a few hundred words or so but the task of collating these words in a dynamic structure is an important cost overall. More generally, in any application in which a small data structure is frequently consulted, inefficiency can significantly degrade performance of the application as a whole.

Compared to the task of managing large sets of strings, different relative performance is to be expected: for example, over only a small number of words of adaptive and balanced structures are unlikely to have the time to reach the equilibrium that in principle might give them an efficiency advantage, presented the results of experiments on a variety of sets of documents. Overall, burst tries are faster than

trees, but by a smaller margin than was observed for larger data sets. Hash tables can be faster again, if sort order is not required, but particularly in comparison to the results for large data sets speed is critically dependent on hash table size. The slowest structure of all is the tree, a striking contrast to the efficiency of this structure on large data sets.

1.2 Search Trees

A standard binary search tree (BST) stores in each of its nodes a key together with a pointer to each of the node's two children. A query for a key q starts at the root; if q is not found in a node, a comparison between q and the node's key determines which branch downwards the query, a new node may be inserted. The performance of a BST depends on the insertion order of keys. The worst case is of O(n) for a BST with n nodes and occurs if the keys are inserted in short order yielding to long sticks. In our large scale practical experiments on string data sets drawn from the web with billions of string occurrences and millions of distinct strings, the performance of a BST was surprisingly good, although a small number of strings in short order at the start of the data led to dramatic degeneration in performance.

A BST should not be used in practice for this task. Considering per-document processing of a text collection. Can reasonably assume that the vast majority of documents will not consist of strings in short order, therefore average logarithmic search cost should dominate. The performance of a BST in this case depends much more on the characteristics of the input. If the collection is skew, as in text where a small percentage of the distinct strings account for a large proportion of the text. Common keys are likely to be observed early in a document. They world then be stored in the upper tree levels. Where only a few comparisons are needed to access them. The common strings tend to be short, further reducing total processing costs. Hence. a BST should be well. Suited to the task of per-document vocabulary accumulation.

The cost of string comparisons during a tree traverse; can be reduced as the search is narrowed: if the query string is known to be lexicographically greater than international "the first six characters can be ignored in subsequent comparisons. However, have found in a range of experiment on large data sets that the additional tests and operations needed to identify how many characters to ignore cost significantly more than the savings, and do not use this technique in any tree structure used in our experiments.

AVL trees and red-black trees are variants of BSTs that reorganize the tree on insertion to maintain approximate balance. They achieve the logarithmic worst case bound by storing some additional information in each node. A single bit in a red-black tree and two bits in a AVL tree. AVL trees and red-black trees guarantee the absence of long sticks but due to the reorganization commonly-accessed keys are not necessarily clustered at the top three levels. On the other hand, for a skew input where most accesses are not insertions the cost of balancing is kept low. The benefits of having a balanced tree structure in contrast to a BST could therefore offset the additional costs that incur at insertion time.

1.3 Sequential Reuse Distance Analysis

Reuse distance analysis can be extremely useful, but also costly. Consequently, many approaches and optimizations have been proposed in the literature, ranging from the full analysis to probabilistic approaches that trade accuracy for performance. To our knowledge, our work is the first attempt at paralyzing reuse distance analysis to leverage multiple cores or nodes in a parallel system to accelerate the analysis. Our approach to parallel reuse distance analysis is compatible with many existing sequential data reference analysis algorithms. The development in this paper utilizes the efficient sequential reuse distance analysis algorithm developed.

1.4 Naive Algorithm

A naïve approach to reuse distance analysis uses a stack data structure to maintain an ordered list of data references. This stack simulates the behavior of an infinite sized, fully associative cache. Initially the stack is empty and data references are pushed onto the head in trace order. When pushing a reference, the stack is traversed starting at the head to determine if there was previous reference to this data element. If the data reference is found. The old instance is removed and pushed onto the head. The distance from the head to the position of the old instance is recorded as the stack, or reuse distance for the new reference. If the data reference is not found, this is the first reference to this data element and it is recorded as having a distance of infinity (corresponding to a compulsory cache miss). For a trace of length N containing M distinct references. N elements must be

processed and at each step an O(M). The space complexity is O(M) since the list can contain at most one entry per distinct data address in reference trace.

1.5 Tree-Based Algorithm

Several tree-based reuse distance analysis algorithms have been developed to reduce the O(M) cost of stack traversal in the naïve algorithm to O(log M). Often a splay tree is used to enhance the efficiency of this structure. These algorithms generally make use of two data structures; a a hash table to store the timestamp of the most recent reference to each data element and a tree that can be used to efficiently determine the number of distinct references since the last time a data element was referenced. Naïve algorithm is the popular tree-based reuse distance analysis algorithm. In this algorithm, a hash table H maps data references to their most recent access timestamp. A binary tree is maintained that holds one entry for each data reference seen thus far and is sorted according to timestamp. Sub tree sums are maintained at each node for the number of elements in the sub tree rooted at the node. Thus the process of counting the distance associated with a data reference is that of traversing the tree from the root to that data reference and accumulating the sum along the way.

The algorithm maintains a balanced binary tree with at most one entry per data reference. Each node contains the reference, the timestamp of the last access. And the sum of the weights in its sub tree. Insertion and deletion are performed using the standard algorithm for balanced binary trees, which ensures that the ordering is preserved and that the tree remains balanced. When performing the distance operation. Find the timestamp of the last access to the current data element using the hash table and then apply the distance calculation algorithm.

1.6. Related Searches

1.6.1. Uniform Cost search

Uniform cost search is a cost based search and it will give an output based on cost. Where the cost is the path cost g(n).

1. It Measures the cost to each node.
2. It is optimal and complete.
3. It can be very slow.

1.6.2. Hill Climbing Search

Expand the node you think is nearest to goal. Where the estimate of distance to goal is h(n). Because it keeps no history, the algorithm cannot recover from failures of its policy. A major problem of hill-climbing is their tendency to become stuck at local data.

1. It Estimates how far away the goal is.
2. It is neither optimal nor complete.
3. It can be very fast.

1.6.3. A* Search

A* search is a combination of uniform cost and hill climbing search. The function of A* search has shown below.

1. Uniform cost = g (n) is the cost to get to a node.
2. Hill Climbing = h (n) is the estimated distance to the goal.
3. A* = f (n) = g (n) + h (n)

Can think of f (n) as the estimated cost of the cheapest solution that goes through node. If the heuristic is optimistic, that is to say, it never overestimates the distance to the goal, then A* is optimal and complete that goal. But some complexity in A* algorithm. A* with worst case is space complexity, but an iterative deepening version is possible.

1.7 Best – First search

The Best-first search uses two sets. One is open dataset (those generated but not yet selected) another one is closed dataset (already selected).

Best-first search progresses by choosing a node with the best f-value in the Open list for expansion. The node is removed from the Open list and each of its children's are generated. For each child node that is generated, best-first search checks the Open and Closed lists to ensure that it is not a duplicate of a previously generated node or, if it is, that it represents a better path than the previously generated duplicate. If that is the case, then the node is inserted into the closed list and a new node is chosen for expansion. This continues until a goal node is chosen for expansion.

Different best-first search algorithms are constructed by choosing different evaluation functions. Frequently the evaluation function is constructed by combining a measurement of the cost incurred by a partial solution path, denoted by the function g(n), and an optimistic estimate of the cost of any path that follows from a node, denoted by the function h(n). The well-known best-first search algorithm uses the evaluation function *f*(n) = g(n) + h(n). In the elimination order search space, the cost of a complete solution path is the width of the corresponding order. It is computed by taking the maximum elimination cost along the solution path, where the cost of a particular elimination is the degree of the vertex being eliminated. A reasonable g-function would thus be the maximum cost incurred in reaching a node. A reasonable h-function would be an optimistic estimate of the maximum cost along some path from the current state to a goal state. These can be combined into the evaluation function f(n) = max g(n), h(n).

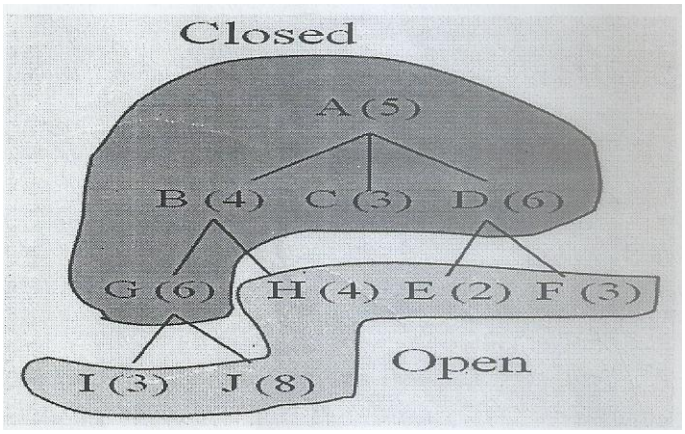
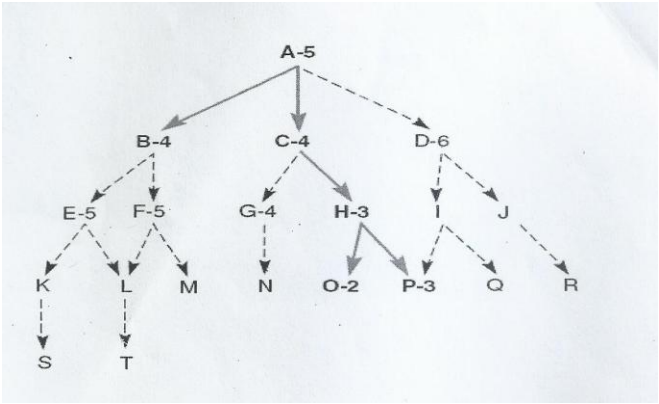


Figure 1.1 Best-First Search Diagram.

1.8 Process Flow of Best-first Search

1. Put the start node *s* on a list called *OPEN* of unexpanded nodes.
2. If *OPEN* is empty exit with failure; no solutions exists.
3. Remove the first *OPEN* node n at which f is minimum (break ties arbitrarily), and place it on a list called *CLOSED* to be used for expanded nodes.
4. Expand node n, generating all its successors with pointers back to n.
5. If any of no's successors is a goal node, exit successfully with the solution obtained by tracing the path along the pointers from the goal back to s.
6. For every successor n' on n:
   I. Calculate f (n')
   II. If n' was neither on OPEN nor on CLOSED, add it to OPEN. Attach a pointer from n' back to n. Assign the newly computed f(n') to node n'.
   III. If n' already resided on OPEN or CLOSED, compare the newly computed f(n) with the value previously assigned to n'. If the old value is lower, discard the newly generated node. If the new value is lower, substitute it for the old (n' now points back to n instead of to its previous predecessor). If the matching node n' resided on CLOSED, move it back to OPEN.
   IV. Go to step 2.
7. Go to step 2.

1.9 Search Flow of Best-First Search



1. **Open=A5; closed=[]**
2. **evaluate A5;open=[B4,C4,D6];closed=[A5]**
3. **evaluate B4;open=[C4,E5,F5,D6];closed=[B4,A5]**

4. **evaluate C4;open=[H3,G4,E5,F5,D6];=closed=[C4,B4, A5]**

5. **evaluate H3;open[O2,P3,G4,E5,F5,D6];closed=[H3,C 4,B4,A5]**

6. **evaluate O2;open=[P3,G4,E5,F5,D6];closed=[O2,H3, C4,B4,A5]**

7. **Evaluate P3; the solution is found!**

Figure 1.2 Best-First Search Flow.

1.10 RFM model

In this thesis, I include the concepts of RFM analysis into sequential pattern mining process. For a given subsequence, each dataset sequence contributes its own regency, frequency, and monetary scores to represent user data importance.

2. LITERATURE REVIEW

This paper shows the practical performance of the following algorithms. Used the data structures which were indicated in the original papers. This paper presents an alternative data structure multi-level link list and apply the heuristic technique to solve shortest path problem. The results indicate that use of this type of data structure helps in improving the performance of algorithms drastically.

Presented major class of heuristic algorithms. The comparison shows that though all these algorithms can be applied to find the shortest path, but should not be used unless there is a real-time, event driven actions are anticipated. The comparison gives us clear idea that best-first nodes. However there may be interesting scenarios that may come out when these algorithms are ascent hill climbing algorithms are not suitable for problems such as shortest path finding. This is due to the fact that there is no assurance of getting final optimal solution for all the cases. Best first and A*algorithms on the other hand ensure optimal solution for all the cases. Best first and A* algorithms on the other hand ensure optimal solution for limited graph size. For larger number of nodes these algorithms not only tend to take more time but the optimality factor may be of concern.

The approach adopt uses weighted heuristic search to find an approximate solution quickly, and then continues the weighted search to find improved solutions as well as to improve a bound on the sub optimality of the current solution. When the time available to solve a search problem is limited or uncertain, this creates an anytime heuristic search algorithm that allows search time and solution quality. Analyse the properties of the resulting Anytime A* algorithm. And consider its performance in three domains; sliding-title puzzles, STRIPS planning, and multiple sequence alignment. To illustrate the generality of this approach, also describe how to transform the memory efficient search algorithm Recursive Best-First search (RBFS) into an anytime algorithm.

The simplicity of the approach makes it very easy to use. It is also widely applicable. Not only can it be used with other search algorithms that explore nodes in best first order, such as RBFS, have shown that it is effective in solving a wide range of search problems. As a rule, it is effective whenever a suboptimal solution can be found relatively quickly using a weighted heuristic, and finding a probably optimal solution takes much longer. That is, it is effective whenever weighted heuristic search is effective. If the weight is chosen appropriately. Have shown that this approach can create a search algorithm with attractive anytime properties without significantly delaying convergence to a provably optimal solution. Conclude that anytime heuristic search provides an attractive approach to challenging search problems. Especially when the time available to find a solution is limited or uncertain.

This paper analyses the problem of UHRs in planning in detail, and proposes a two level search framework as a solution. In Greedy Best-First Search with Local Exploration (GBFSLE), a local exploration is started from within a global GBFS whenever the search seems stuck in UHRS. Two different local Random walk Search (LRW). The two new planners LAMA-2011. Both are shown to yield clear improvements in terms of both coverage and search time on standard international planning Competition benchmarks, especially for domains that are proven to have large or unbounded UHRs.

While local exploration has been investigated before in the Context of local search planners, it also serves to facilitate Escaping from UHRS for greedy best-first search. The new Framework of GBFS-LF, GBFS with local exploration, has been tested successfully in two different realizations, adding Local greedy best-first search in GBFS-LS and random walks In GBFS-LRW.

This paper proposed and evaluated the power of best-first search over AND/OR search spaces in graphical models. The main virtue of the AND/OR representation is its sensitivity to the structure of the graphical model, which can translate into significant time savings. Indeed, in recent years depth-first AND/OR Branch-and-Bound algorithms were shown to be very effective when exploring such search spaces, especially when using caching. Since best-first strategies are known to be superior to depth-first when memory is utilized, exploring the best-first control strategy is called for. In this paper introduce two classes of best-first AND/OR search algorithms: those that explore a context-minimal AND/OR search graph and use static variable orderings, and those that use dynamic variable orderings but explore an AND/OR search tree. The superiority of the best-first search approach is demonstrated empirically on various real-world benchmarks.

This paper considered Weighted Best First (WBF) search schemes, popular for path-finding domain, as approximations and as anytime schemes for the MAP task. Author demonstrate empirically the ability of these schemes to effectively provide approximations with guaranteed sub optimality and also show that as anytime scheme, Depth-First Branch and-Bound.

In this paper extended advanced best-first scheme for graphical model into a weighted scheme and evaluated its performance in comparison with a highly competitive Branch and Bound scheme. Our empirical results show that weighted best-first is valuable in providing relatively fast solutions together with sub optimality bounds. Demonstrated that weighted best-first search schemes should definitely be included in the set of good optimization schemes for solving MPE/MAP tasks. The weight mechanism can mitigate the memory/time trade off in a useful way that can be harnessed into an anytime scheme that not only improves with time, but can also guarantee its level of sub optimality.

This paper particularly, focused on an approach which distributes and schedules work among processors based on a hash function of the search state. Use this approach to parallelize the A* algorithm in the optimal sequential version of the Fast Downward planner. The scaling behavior of the algorithm is evaluated experimentally on clusters using up to 128 processors, a significant increase compared to previous work in parallelizing planners. This approach scales well, allowing us to effectively utilize the large amount of distributed memory to optimally solve problems which require hundreds of gigabytes of RAM to solve. Also show that this approach scales well for a single, shared-memory multicore machine.

In this paper, above mention author compare different approaches to parallel best-first search in a shared-memory setting. Present a new method, PBNF that uses abstraction to partition the state space and to detect duplicate states without requiring frequent locking. PBNF allows speculative expansions when necessary to keep threads busy. We identify and fix potential live lock conditions in our approach, proving its correctness using temporal logic. Our approach is general, allowing it to extend easily to suboptimal and anytime heuristic search. In an empirical comparison on STRIPS planning, grid path finding, and sliding tile puzzle problems using 8-core machines, we show that A*, weighted A* and Anytime weighted A* implemented using PBNF yield faster search than improved versions of previous parallel search proposals.

The objective of this paper is to present a comprehensive methodology to discover the knowledge for selecting targets for direct marketing this study expanded RFM model by including two parameters, time since first purchase or length of customer relationship and cost of a customer. Authors in this paper first review the CRM concept and RFM model and next propose modified model. In the empirical study Authors examine a case study, insurance study. Authors cluster the insurance customer with k-means algorithm. The result show that the modified model is better than base RFM model.

In this study authors examine the RFM model and authors tried to make the model more efficient variables time of relationship and associated costs added to the model. Authors used association rules to compare two models. The result was that the new model works better than the RFM.

These methods provide a lot of opportunities in the market sector. This paper deals with mining algorithms and methods (especially RFM analysis) and their use in Six Sigma methodology, especially in DMAIC phases. DMAIC stands for Define, Measure, Analyse, Improve and Control. Our research is focused on improvement of Six Sigma phases (DMAIC phases). With implementation of RFM analysis (as a part of Data Mining) to Six Sigma (to one of its phase). We can improve the results and change the Sigma performance level of the process. We used C5.0, QUES, CHAID and Neural Network algorithms. The results are in proposal of selected Data Mining methods into DMAIC phases.

The RFM model provides an effective measure for customers' consumption behavior analysis, where three variables, namely, consumption interval, frequently, and money amount are used to quantify a customer's loyalty and contribution. Based on the RFM value, customers can be clustered into different groups and the group information is very useful in market decision making. However, most previous works completely left out important characteristics of purchased products, such as their prices and lifetimes, and apply the RFM measure on all of a customer's purchased products. This renders the calculation of the RFM value unreasonable or insignificant for customer analysis. In this paper, we propose a new framework called GRFM (for group RFM) analysis to alleviate the problem. The new measure method takes into account the characteristics of the purchased items so that the calculated the RFM value for the customers are strongly related to their purchased items and can correctly reflect their actual consumption behavior. Moreover, GRFM employs a constrained clustering method PICC (for Purchased Items-Constrained Clustering) that could base on a cleverly designed purchase pattern table to adjust original purchase records to satisfy various clustering constraints as well as to decrease re-clustering time. The GRFM allows a customer to belong to different clusters, and thus to be associated with different loyalties and contributions with respect to different characteristics of purchased items. Final, the clustering result of PICC contains extra information about the distribution status inside each cluster that could help the manager to decide when is most proper to launch a specific sales promotion campaign. Our experiments have confirmed the above observations and suggest that GRFM can play an important role in building a personalized purchasing management system and an inventory management system.

RFM (Recency, Frequency and Monetary) model has been widely applied in many practical areas in a long history, particularly in direct marketing. By adopting RFM model, decision makers can effectively identify valuable customers and then develop effective marketing strategy. This paper aims to provide a comprehensive review on the application of RFM model. In addition, this paper depicts the definition and the scoring scheme of RFM and summarizes how RFM model has been effectively applied in a wide variety of areas. Furthermore, this paper presents the advantages and disadvantages of the RFM models are also exploited. Finally, this paper describes the extended RFM model via a presentation of how RFM combines with other variables and models.

3. PROBLEM DEFINITION AND METHODOLOGY

3.1. Problem

Best-first search is admissible on max-cost problem. There are multiple equivalent definitions of search that suggest unique ways of thinking about the problem. Our efforts are focused on finding best search data list to user required results in terms of optimal data cost elimination orders. Eliminating a min cost from a data server is defined as the process of adding a cost between every pair of the data neighbors that are not already adjacent, then removing the vertices and all incident cost from the storage. A data eliminating order is a total order over the data's in storage. An algorithm that finds the exact search of a server can dramatically improve the performance of exact inference and thereby increase the size of problems that can be solved in practice. This search space has many duplicate data, and propose using algorithms, like best-first search, that detect and eliminate all of them. The version of best-first search is most often used in practice, which applies to shortest-path problems.

3.2. Problem Solving

Obviously the user wants to use heuristic search, but for some domains (as we'll see later) good heuristics are hard to produce. If not, there are memory and time considerations.

BFS and the like are guaranteed to find short path (data)s, but use a little of memory.

DFS is much faster, but isn't guaranteed to find a solution and takes lot of memory. Even for heuristic search sometimes it does the equivalent of DFS on the heuristic value.

Best first search is a type of graph search algorithm. Here the nodes are expanded one at time by choosing lowest evaluation value. This evaluation value is a result of heuristic function giving a measure of distance to the goal node. For typical applications such as shortest path (data) problems, the evaluation function will be accurate as it accounts for distance or an absolute value. Best first search is combination of breadth and depth first search. Depth first search has an advantage of arriving at solution without computing all data's in the storage server, whereas breadth first arriving at solution without computing all data's in the storage server, whereas breadth first arriving at solution without search ensured that the process does not get trapped. Best-first search, being combination of these two, permits switching between path (data).

At every stage the data's among the generated ones, the best suitable data is selected for further expansion, may be this data belong to the same level or different, thus can toggle between depth-first and breadth-first. This method involves search result to avoid duplication, and also requires two separate lists for processing. OPEN list keeps the data whose heuristic values are determined, but yet to be expanded. CLOSE list have the data which have been already checked, further these data are kept in this list to ensure no duplications. It implies that the OPEN list has the data's which need to be considered for further processing and the entries in CLOSE list indicate the data's which may not be re-required in further steps.

3.3 Methodology

OPEN = (initial state)

While OPEN is not empty or until a goal is found

Do

1. Remove the best node from OPEN, call it n.
2. If n is the goal state, back trace path to n (through recorded parents) and return path.
3. Create n' s successors.
4. Evaluate each successor, add it to OPEN, and record its parent.

Done.

3.4 RFM Model

The next step involves determining values and scoring RFM variables and using them as inputs of clustering algorithm. Output of the RFM model consists of three fields of each customer: frequently, recently and monetary. RFM model is proposed by Hughes, and has been used in direct marketing for several decades. This model identifies customer behavior and represents customer behavior characteristics by three variables:

1. Recency of the last purchase (access) which refers to the interval between latest customer purchase and time analysis of customer data.
2. Frequency of the purchase (access) a which refers to the number of transactions in a particular period.
3. Monetary value of the purchase (access) which refers to consumption amount (like data size) in a particular period.

RFM model can be used in different areas by different people: Therefore, RFM can mean different things to different people. Classic RFM ranks each data based on valuable against other customers (access) and RFM Score will be assigned to each and every data.

In my thesis I have used this technique to reduce the memory space based on the needs.

4. CONCLUSION AND FUTURE WORK

4.1 Conclusion

I have presented major class of heuristic algorithms. Though all the algorithms can be applied to find the shortest path, but should not be used unless there is a real-time, event driven action are anticipated. The comparison gives us clear idea that best-first search algorithms are very well suitable when goal node cannot be reached from all nodes. However there

may be interesting scenarios that may come out when these algorithms are applied with different data structures. The results clearly indicate that steepest ascent hill climbing algorithms are not suitable for problem such as shortest path finding. This is due to the fact that there is no assurance of getting final optimal solution for all the cases. Best first algorithms on the other hand ensure optimal solution for limited graph size. For larger number of nodes these algorithms not only tend to take more time but the optimality factor may be of concern.

4.2 Future Work

The primary goal of future work will be to increase the size of problems that can be solved in a reasonable amount of time. One approach for doing this is to continue and investigate algorithms that eliminate all duplicate nodes, specifically disk-based search. These methods greatly increase the amount of available memory by utilizing external disk-based search. These methods greatly increase the amount of available memory by utilizing external disk storage while limiting the added costs in terms of running time.

Another approach for solving larger problems involves constant-space algorithms. The idea behind these methods is to use all the available memory to eliminate as many duplicates as possible. This can be done with a memory-bounded version of best-first search, or by adding a transposition table to depth-first branch-and-bound. Finally, since anytime algorithms are useful in some contexts it will be interesting to evaluate anytime variants of the algorithms developed in this paper. This can be done by finding a min-fill path from interior search nodes.

4. REFERENCES

[1] "Comparison of various heuristic Search techniques for finding Shortest path "Mr. Girish p potdar, dr.r.c. thool. Vol. 5, No.4, July 2014.

[2] "Anytime Heuristic Search" Eric A. Hansen, Rong Zhou Submitted 05/06; published 03/07.

[3] "Adding Local Exploration to Greedy Best-First Search in Satisficing Planning" Fan Xie and Martin Muller and Robert Holte Proceedings of the Twenty-second International Conference on Automated Planning and Scheduling (ICAPS-2012)

[4] "Best-First AND/OR Search for Graphical Models" Radu Marinescu and Rina Dechter. 2007, Association for the Advancement of Artificiallntelligence (www.aaai.org).

[5] "Weighted Best First Search for Map" Natalia Flerova, Radu Marinescu, Rina Dechter, Thayer, and Rum (2010).

[6] "Scalable, Parallel Best-First Search for Optimal Sequential Planning" Akihiro Kishimoto, Alex Fukunaga, Adi Botea and Hansen (2007).

[7] "Best-First Heuristic Search for Multicore Machines" Ethan Burns, Sofia Lemons, Wheeler Ruml Journal of Artificial Intelligence Research 39 (2010) 689-743.

[8] " Customer Segmentation based on Modified RFM Model in the Insurance Industry" Reza Allahyari Soeini Ebrahim Fathalizade SIMULATION Online First, published on October 22, 2009.

[9] "Six Sigma Methodology with Recently, Frequently and Monetary Analysis Using Data Mining" Andrej Trnka Received 8 June 2005; Revised 20 November 2006; Accepted 22 November 2006.

[10] "Group RFM analysis as a novel frame work to discover better customer consumption behavior" Hui-Chu Chang, Hsiao-Ping Tsai vol. 4007, pp. 109-120. Springer, Heidelberg (2006).

[11] "A review of the application of RFM mode" Jo-Ting Weil, Shih-Yen Lin and Hsin-Hung Wu CA, pp. 39-48. SIAM, Philadelphia, PA (2002).

[12] D. Dreyfus, (1967) "An appraisal of some shortest path algorithms", Journal of the Operations Research Society of America, Vol. 17 Issue 3, pp 395-412.

[13] A.V. Goldberg, (2001) "A simple shortest path algorithm with linear average time", In proceeding 9th ESA, Lecture notes in computer science LNSC 2161. Pp 230-261.

[14] B.V. Cherkassy, A.V. Goldberg, T.Radzik, (1996) "Shortest Path Algorithms: theory and experimental evaluation", Mathematical Programming, 73 (2) pp 129-74.

[15] J.W. Lark, C.C. White III, K. Syverson., (1995) "A best first search algorithm guided by a set- valued

heruistic", IEEE Transactions on Systems, Man, and Cybernetics, Vol. 25, pp 1097-1101.

[16] R.K. Ahuja, K. Mehlhorn, J.B. Orlin and R.E. Tarjan, (April 1990) "Faster algorithms for shortest path algorithms", Journal of the Association for Computing Machinery, Vol. 37, No.2, pp 213-223.

[17] D.P. Bertekas, (1991) "The auction algorithms for shortest paths". SIAM J. Opt, Vol. 1, pp 425-447.

AUTHORS

[1]M.Sinthiya

Msc, Mphil (computer science)

Rajah Serfoji Government Arts College

[2] Dr.M. Chidambaram Msc, Mphil, MBA, PhD (Asst.Proff)
Rajah Serfoji Government Arts College

APPENDIX

Find Shortest Distance

As an illustration of Dijkstra's procedure, let us find the distance from vertex B to vertex G in the digraph shown in Figure A1.

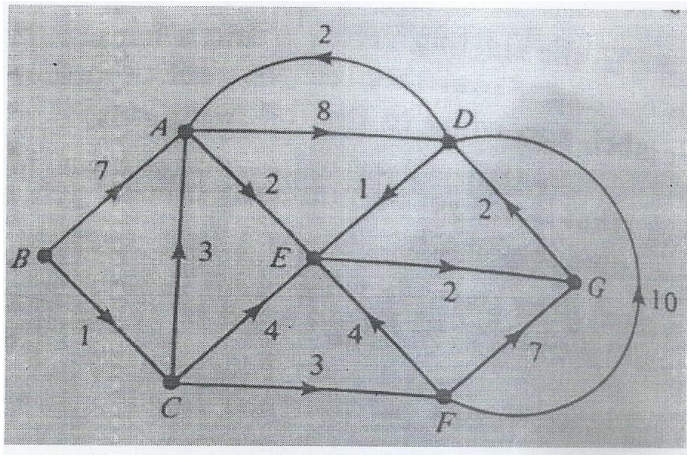


Figure A1. Simple weighted graph

Input: A search graph problem with cost on the arcs

Output: The minimal cost path from start node to a goal node.

a) Expand the cheapest next node.
b) Cost is the vertex cost g (n)
c) Expand the node you think is nearest to goal
d) The estimate of distance is h (n) counted path distance
e) The estimate the next vertex is not avail distance is assigned infinity
f) Measures the cost to each node.
g) Is optimal and complete.
h) Can very slow.
i) Can combine them to create an optimal and complete algorithm.
j) $f(n) = g(n) + h(n)$

A Heuristic is a function that, when applied to a state, returns a distance that is an estimate of the goal.

In other words, the heuristic tells us approximately how far the state is from the goal state. Note said "approximately". Distance might underestimate or overestimate the merit of a state. But for reasons which will see, heuristics that only underestimate are very desirable, and are called admissible.

Vertex = v

Distance = d

Path p = Add Next node

Path-cost = Distance to node in miles.

Minimum = Minimum time, least fuel.

Path-cost = Number of node moved.

Minimum = Least time to solve.