Shortest Path Algorithms Techniques in Case retrieval phase of case-based Reasoning

Dr. Gundeep Tanwar

Associate Professor, BRCM College of Engineering, Bhiwani, Haryana

Abstract— **This paper discuss the shortest path algorithm in the case retrieval phase of the case-based reasoning approach. The case-based reasoning approach is mainly used in the problem solving phenomena with utilization of past problem solving experiences. The case retrieval phase is concerned with finding the similar cases in the case base. this phase makes the impact on the performance of the case-based reasoning system. There exists multiple case retrieval phases, Shortest paths, or close to shortest paths, are commonly used in everyday situations. The paper reviews the various algorithms available for the problem. One of the famous technique Dijkstra's algorithm solves the single-source shortest path problem on any directed graph in O(m+nlogn) worst-case time when a Fibonacci heap is used as the frontier set data structure**.

Keywords— **Case-based reasoning, case retrieval, Shortest path algorithm.**

I. INTRODUCTION

Case-based reasoning is a problem solving paradigm that in many respects is fundamentally different from other major AI approaches. Instead of relying solely on general knowledge of a problem domain, or making associations along generalized relationships between problem descriptors and conclusions, CBR is able to utilize the specific knowledge of previously experienced, concrete problem situations (cases). A new problem is solved by finding a similar past case, and reusing it in the new problem situation. A second important difference is that CBR also is an approach to incremental, sustained learning, since a new experience is retained each time a problem has been solved, making it immediately available for future problems. The CBR field has grown rapidly over the last few years, as seen by its increased share of papers at major conferences, available commercial tools, and successful applications in daily use.

The description of CBR principles, methods, and systems is made within a general analytic scheme. Other authors have recently given overviews of case-based reasoning. To solve a new problem, the CBR remembers previous similar situation and reuses information and knowledge of that situation. The CBR paradigm covers a range of different methods for organizing, retrieving, utilizing and indexing the knowledge retained in past cases. Cases may be kept as concrete experiences, or a set of similar cases may form a generalized case. Cases may be stored as separate knowledge units, or splitted up into subunits and distributed within the knowledge structure. Cases may be indexed by a prefixed or open vocabulary, and within a flat or hierarchical index structure. The solution from a previous case may be directly applied to the present problem, or modified according to differences between the two cases. The matching of cases, adaptation of solutions, and learning from an experience may be guided and supported by a deep model of general domain knowledge, by more shallow and compiled knowledge, or be based on an apparent, syntactic similarity only.

CBR methods may be purely self-contained and automatic, or they may interact heavily with the user for support and guidance of its choices. Some CBR method assume a rather large amount of widely distributed cases in its case base, while others are based on a more limited set of typical ones. Past cases may be retrieved and evaluated sequentially or in parallel. Actually, "case-based reasoning" is just one of a set of terms used to refer to systems of this kind.

II. CASE RETRIEVAL PHASE

A CBR tool should support the four main processes of CBR: retrieval, reuse, revision and retention. A good tool should support a variety of retrieval mechanisms and allow them to be mixed when necessary. In addition, the tool should be able to handle large case libraries with retrieval time increasing *linearly (at worst) with the number of cases*

Case retrieval is a process that a retrieval algorithm retrieves the most similar cases to the current problem. Case retrieval requires a combination of search and matching. In general, two retrieval techniques are used by the major CBR applications: nearest neighbor retrieval algorithm and inductive retrieval algorithm.

Nearest-Neighbor Retrieval

Nearest-neighbor retrieval is a simple approach that computes the similarity between stored cases and new input case based on weight features. A typical evaluation function is used to compute nearest-neighbor matching [Kolodner, 1993] as shown in Figure 1:

$$similarity(Case_I, Case_R) = \frac{\sum_{i=1}^{n} w_i \times sim(f_i^I, f_i^R)}{\sum_{i=1}^{n} w_i}$$

Figure 1 Nearest-neighbor evaluation function

Where w_i is the importance weight of a feature, *sim* is the similarity function of features, and f_i^I and f_i^R are the values for feature *i* in the input and retrieved cases respectively.

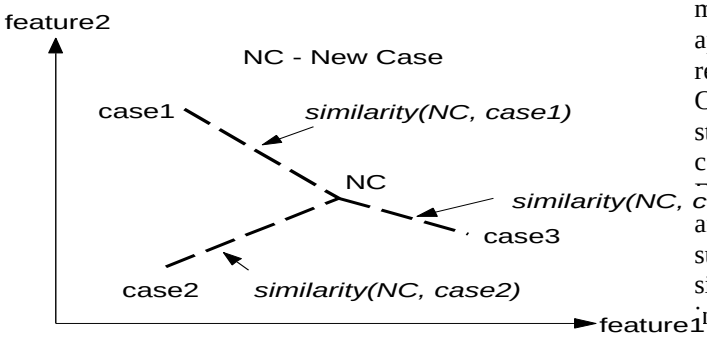

Figure 2 displays a simple scheme for nearest-neighbor matching. In this 2-dimensional space, *case3* is selected as the nearest neighbor because *similarity*(*NC, case3*)> *similarity*(*NC, case1*) and *similarity*(*NC, case3*)> *similarity*(*NC, case2*).

III. REVIEW OF LITERATURE

Pan et al. (2007) presented a novel algorithm for automatically mining a high-quality case base from a raw case set that could preserve and sometimes even improve the competence of case-based reasoning. In this paper, they analyzed two major problems in previous case-mining algorithms. The first problem was caused by noisy cases such that the nearest neighbor cases of a problem may not provide correct solutions. The second problem was caused by uneven case distribution, such that similar problems may have dissimilar solutions. To solve these problems, they developed a theoretical framework for the error bound in case-based reasoning, and proposed a novel case-base mining algorithm guided by the theoretical results that returned a high-quality case base from raw data efficiently. They supported their theory and algorithm with extensive empirical evaluation using different benchmark data sets.

Wang et al. (2007) presented a new cost estimation concept based on the case-based reasoning (CBR) approach instead of a traditionally intuitive estimation method. In CBR model, two retrieval techniques, 'Inductive Indexing' and 'Nearest Neighbor', were then applied to retrieve relevant cases from the knowledge-based database. Two of the most common types of Taiwan historical buildings were tested to explore the restoration cost implications. The result revealed that the CBR solution could effectively predict the actual restoration cost, solve order change problems, and reduce the budget review time. These applications were also useful for many other countries, especially for those seismic belt regions, that were facing similar problems regarding historical building restoration.

Gomes et al. (2004) developed a system capable of providing these requirements. It had a central knowledge base that could be used through Case-Based Reasoning. The knowledge base integrated a common ontology called WordNet, providing classification for software objects. This paper focuses on the retrieval of design models using the combination of WordNet and Case-Based Reasoning. They also presented a retrieval example, and experimental work showing the performance of the retrieval and ranking mechanisms.

Velandia et al. (2008) developed proposed most CBR retrieval algorithms which employed a modified version of the nearest neighbour rule that used a distance function as similarity measure, which in turn depends upon the attribute type. The application of moment-based retrieval used in image recognition for CBR retrieval is studied in this paper. Comparison with the classical retrieval algorithms that used standard distance measures showed that low-order geometric, central, and Legendre moments retrieve the same cases as the Euclidean distance does, whereas high-order geometric, central, and Legendre moments retrieved different cases. It was suggested that there was not a single distinguished approach to similarity in CBR, rather CBR systems should allow the integration of different approaches to similarity and the selection of different concepts.

The use of shorter paths occurs naturally when traveling between two locations, whether this is travel from one room to another, from one street address to another, or from one city to another. Taking a long path typically makes no sense, since doing so results in time being wasted. Thus, shorter paths are preferred for reasons of efficiency. To achieve the greatest efficiency when traveling between two points, it is necessary to take a path that is shortest among all possible paths; that is, the shortest path. Generally speaking, a shortest path is one of minimal cost.

The problem of computing shortest paths commonly arises when the most cost-efficient route through a transportation or communication network needs to be found. In the case of transportation, cost may be represented by a combination of factors, including distance traveled, time spent, fuel used, tolls paid, or many other factors.

Basic Terminology:-

The exact definition being used for cost depends on the specific problem being solved. While shorter paths tend to be used naturally, determining truly shortest paths allows more efficient use of networks. Solving shortest paths by plain intuition is not always guaranteed to obtain the correct result. The truly shortest path, or that of minimum cost, is not always the most obvious choice.

For example, consider finding the shortest path in order to minimize the time spent traveling between two locations in a city. Here cost is measured in terms of the time spent traveling. The shortest path may require taking a detour in order to avoid traffic congestion. Such a path can be completely different from the path that is shortest in terms of distance traveled. Even with cost defined as distance traveled the correct choice of shortest path may be counter-intuitive. Furthermore, large shortest path problems are typically too complex to solve accurately by hand. By computing shortest paths, rather than using intuition, a correct result can always be obtained. Shortest path problems in general are described using the concept of a graph may be either directed or undirected. The edges in an undirected graph have no direction associated with them, and can be thought of as allowing travel in both directions.

In contrast, the edges in a directed graph have an associated direction, which can be thought of as specifying the direction of travel. Think of edges in a directed graph as being one-way, and edges in an undirected graph as two-way. The edges of a graph can be weighted, in which case each edge has an associated cost. In the case of a transportation network, this cost may be the distance along a road between two vertices.

Shortest path problems are represented using directed graphs, since the cost from one vertex to another may be different in the

opposite direction. The edges in a graph form paths connecting vertices. Any such path similarly has an associated cost (or distance), which corresponds to the sum of costs of edges along the path. The existence of alternative paths between a pair of vertices in a graph

IV. SEARCH ALGORITHMS

One possible approach to solving shortest path problems would be to pre-calculate and store the shortest path from every node to every possible other node, which would allow us to answer a shortest path query in constant time. Unfortunately the required storage size and computation time grows with the square of the number of nodes. With realistic road networks in mind this processing would take years if not decades and be virtually impossible to store. Hence to overcome this problem we require real time search techniques. From previous studies we know that the implementation of labelling algorithms is the fastest for one-to-one searches. Two aspects are particularly important to the shortest path algorithms discussed in this project:

1. the strategies used to select the next node to be visited during a search, and

2. the data structures utilized to maintain the set of previously visited nodes.

A number of data structures can be used to manipulate the set of nodes in order to support search strategies. These data structures include arrays, singly and doubly linked lists, stacks, heaps, buckets and queues. Detailed definitions and operations related to these data structures are standard knowledge and are well documented. Past research has concentrated mainly on the issue of data structures, which can be manipulated and bounded to form clever techniques in creating priority queues for selecting nodes to be scanned. A good example of this is the Dijkstra implementation with double buckets. In a labeling algorithm, the number of visited nodes during a search is a good indication of the size of the search space. This means that a search strategy which visits fewer nodes during a search is generally more efficient in terms of processing speed. The number of nodes visited depends on the depth d (i.e. the number of arcs on the optimal path) of the destination from the origin, and the branching factor b. For a 'best first search' the number of nodes explored during a search is of the order O(bd). This exponential growth in the number of explored nodes is known as "combinatorial explosion" and is the main obstacle in computing shortest paths in large networks. (Note that even though Dijkstra's algorithm is polynomial in the number of nodes n in the graph, this bound is no restriction on how the number of nodes visited varies with d). For general search this exponential growth with depth makes many problems unsolvable on current hardware, as memory is soon exhausted and a solution may take an unreasonable time to compute. These effects can be lessened by using artificial intelligence (heuristic type) techniques which will be discussed later. However let us first define and implement Dijkstra's labeling algorithm.

Dijkstra's Naive Implementation:

Dijkstra's labeling method is a central procedure in shortest path algorithms. The output of the labeling method is an out-tree from a source node s, to a set of nodes L. An out-tree is a tree originating from the source node to other nodes to which the shortest distance from the source node is known. This out-tree is constructed iteratively, and the shortest path from s to any destination node t in the tree is obtained upon termination of the method. Three pieces of information are required for each node i in the labeling method while constructing the shortest path tree:

- the distance label, d(i),
- the parent-node/predecessor p(i),
- the set of permanently labeled nodes L.

The distance label d(i) stores an upper bound on the shortest path distance from s to i, while p(i) records the node that immediately precedes node i in the out-tree. If a node has not yet been added to the out-tree, it is considered 'unreached'. Normally the distance label of an unreached node is set to infinity. When we know that the shortest path from node s to node i is also the absolute shortest path, then node i is called permanently labeled. When further improvement is expected to be made on the distance from the origin to node i, then node i is considered only temporarily labeled. It follows that d(i) is an upper bound on the shortest path distance to node i if node i is temporarily labeled, and d(i) represents the final optimal shortest path distance to node i if the node is permanently labeled . By iteratively adding a temporarily labeled node with the smallest distance label d(i) to the set of permanently labeled nodes L, *Dijkstra's algorithm guarantees optimality.*

One advantage with Dijkstra's labeling algorithm is that the algorithm can be terminated when the destination node is permanently labeled. Most other algorithms guarantee optimal shortest paths only upon termination when the entire shortest path tree has been explored.

Symmetrical Dijkstra Algorithm:-

Pohl adapted Dijkstra's shortest path algorithm to decrease the size of the search space. Pohl's algorithm was the first to use a bi-directional search method. This algorithm consists of a forward search from an origin node to the destination node and a backwards search from the destination node to the origin node. This was done in an attempt to reduce the search complexity to O(bd/2) compared to O(bd) as with Dijkstra's algorithm. This search method assumes that the two searches grow symmetrically and will meet in some middle area. Sometimes this might not be the case, and as a worst-case scenario this might instead become two O(bd) searches.

The Symmetrical or Bi-directional Dijkstra's algorithm by Pohl grows two search trees, one from the origin, giving a tree spanning a set of nodes LF for which the minimum distance/time from the origin is known, and a second from the destination that gives a tree spanning a set of nodes LB for which the minimum distance/time to the destination is known. We iteratively add one node to either LF or LB until there exists an arc crossing from LF to LB.

Like Dijkstra's algorithm Pohl's bi-directional search chooses the node with the smallest cost label to label permanently. By selecting the new permanently labeled node from either the forward or backward phases we maintain the Dijkstra criterion required for optimality.

A* Search:-

So far we have examined search techniques that can be generalized for any network (as long as it does not contain negative length cycles). However the physical nature of real road networks motivates investigation into the possible use of heuristic solutions that exploit the near-Euclidean network

structure to reduce solution times while hopefully obtaining near optimal paths. For most of these heuristics the goal is to bias a more focused search towards the destination. As we shall see, incorporating heuristic knowledge into a search can dramatically reduce solution times. When the underlying network is Euclidean or approximately Euclidean as is the case of road networks, then it is possible to improve the average case run time of the Dijkstra and Symmetrical Dijkstra algorithms. This is usually at the expense of optimality; solutions are now not guaranteed to be the best. Typically when solving problems on such networks the inherent geometric information is ignored by algorithms that are directly based or variations on Dijkstra's labeling algorithm. The A* algorithm by Hart and Nilsson formalized the concept of integrating a heuristic into a search procedure. Instead of choosing the next node to label permanently as that with the least cost (as measured from the start node), the choice of node is based on the cost from the start node plus an estimate of proximity to the destination (a heuristic estimate). To build a shortest path from the origin s to the destination t, we
use the original distance from s accumulated along the edges (as in Dijkstra's algorithm) plus an estimate of the distance to t. Thus we use global information about our network to guide the search for the shortest path from s to t. This algorithm places more importance on paths leading towards t than paths moving away from t. In essence the A* algorithm combines two pieces of information:
1. the current knowledge available about the upper bounds (given by the distance labels d(i)), and
2. an estimate of the distance from a leaf node of the search tree to the destination.
There are several ways to estimate the lower bound from a leaf node in the search tree to the destination node. These estimations are carried out by so called "evaluation" functions. The closer this estimate is to a tight lower bound on the distance to the destination, the better the quality of the A* Search. Hence the merits of an A* search depends highly on the evaluation function h(i,j). There are two main evaluation functions used in the A* search. A true lower bound between two points is the length of a straight line between those two points (i.e. the Euclidean distance):

$$h_e(i,t) = \sqrt{((x(i)-x(t))^2 + (y(i)-y(t))^2)}$$

where x(i), y(i) and x(t), y(t) are the coordinates for node i and the destination node t respectively. The other commonly used evaluation function is the Manhattan distance h. In this case the estimated lower bound distance is the sum of distance in the x and y coordinates.

$$hm(i,t) = | x(i)-x(t) | + | y(i)-y(t) |$$

The Manhattan distance is not the true lower bound between two points and hence will typically yield non-optimal results. By using time as a measure of cost, the network becomes near-Euclidean. This is because of the varying speeds of roads in the network. Roads of similar lengths might have different times associated with using those roads. If the network is not strictly Euclidean but near-Euclidean then our selection criteria for the next node to label permanently will not yield optimal results. By using the A* search, the shortest path tree should now grow towards t (unlike Dijkstra's algorithm where the tree grows approximately radially). As before, the search for the shortest path is terminated as soon as t is added to the shortest path tree. Earlier we discussed the problem of combinatorial explosion with a blind search time complexity in the order of O(bd). With A* search this is reduced to O(bed) where be is the effective branching factor. The A* search reduces the search space by reducing the number of node expansions. Although A* is still susceptible to the problem of combinatorial explosion, it decreases the effect by reducing the size of the base in the complexity term.

Weighted A* Search:-

By choosing an appropriate multiplicative factor we can increase the contribution of the estimated component in calculating the label of a vertex (i.e. increase the contribution of the evaluation function). From an intuitive standpoint this corresponds to further biasing the forward search towards the destination and the backward search towards the origin. The heuristic is parameterized by the multiplicative factor termed the "overdo" parameter used to weight the evaluation function. This modification will generally not yield optimal paths, but we would expect it to further reduce the search space. The aim is to find an "optimal" multiplicative or overdo factor for which the running time is significantly improved while the solution quality is still acceptable. Thus there will be an empirical time/performance trade-off as a function of the overdo parameter.

Radius Search

To eliminate or minimize the effects of combinatorial explosion we need to adopt a search technique similar to the way humans approach navigation problems. So far we have not implemented any intelligence within a search which can filter out roads that are less likely to be traveled on. This type of intelligence requires some form of historical knowledge about the network. Since the road network does not change very often it is possible to calculate auxiliary information in a pre-processing step. Perhaps the most obvious way to classify the roads in the network is to identify the class of each road (i.e. motorways, highways, local roads etc), and then to exploit these classes in the search. This is similar to the way humans approach routing problems and is known as Hierarchical Search. Hierarchical methods offer the prospect of greatly reducing the size of any search by simplifying the search through a series of simplified levels, where each of these levels is an abstraction of the previous level. These abstractions reduce the overall size of the search space that an algorithm addresses and thus the complexity of any search is reduced. For route finding, hierarchical levels are constructed in which higher speed roads are placed higher up in the hierarchy. However by introducing these arbitrary hierarchies the path optimality is often lost. The hierarchical algorithm uses a discrete number of hierarchy levels. A Radius search is a hierarchical search with a continuous range of hierarchy levels. A Radius search takes advantage of the fact that the fastest path between two junctions is more likely to use a highway than a local road, especially if the two junctions are far apart. In this method each node i has an associated radius r(i). Before we consider how r(i) is calculated, we first examine how radii can be used to restrict a search. When looking for a shortest path from s to t, a node i is considered as a possible node to include in the search only if s

or t lies inside a circle of radius r(i) centered at node i. If both distances are greater than the node radius, the node is simply ignored. For any given origin and destination node, we can immediately simplify the network by removing all the nodes (and associated arcs) whose radii do not encircle the origin or destination nodes. The radius search is not a search algorithm by itself, but an independent mechanism of reducing search complexity. Hence the radius concept can be used in conjunction with any search algorithm. The effectiveness of the Radius search depends on the way we calculate.

V. CONCLUSIONS

Case-based reasoning systems are an alternative, in many situations, to rule-based systems. In many domains and processes, referring to cases as a means of reasoning can be an advantage due to the nature of this type of problem solving. Mostly the case-based reasoning system uses the k-NN algorithm. By exploiting the physical structure of road networks, the A* algorithm is able to bias its search towards a goal and reduce the search space. By using the concept of radii as a measure of importance of nodes, we are able to incorporate pre-processing within our shortest path algorithm to further restrict the search space. This dramatically reduces the search complexity in terms of the run time performance while still maintaining an acceptable level of inaccuracy.

REFERENCES

[1]. Aamodt, A. & Plaza, E.. Case-Based Reasoning: Foundational Issues, Methodological Variations, and System Approaches. 7(1), (pp. 39-59). AI Communications, 1994

[2]. Ian Watson & Farhi Marir, ―Case-Based Reasoning: A Review‖ The Knowledge Engineering Review, Vol.9 No.4, pp. 1-34, 1994.

[3]. David B. Leake, Case-Based Reasoning: Experiences, Lessons, and Future Directions. Chapter 1, Menlo Park: AAAI Press/MIT Press, 1996.

[4]. Nick Cercone, ―Rule-Induction and Case-Based Reasoning: Hybrid Architectures Appear Advantageous‖, IEEE Transactions on Knowledge and Data Engineering, Vol. 11, No. 1, January 1999.

[5]. Zhi-Ying Zhang, ―A Model for Retrieval base on ANN and Nearest Neighbor Algorithm‖ Proceeding of Seventh International Conference on Machine Learning & Cybernetics, 142-147, 2008.

[6]. Jin Qi, Jie Hu, ―A new adaptation method based on adaptability under k-nearest neighbors for case adaptation in case-based design‖, Expert Systems with Applications Volume 39, Issue 7, 1 June 2012, Pages 6485–6502

[7]. Reyes Pavón Rial and Rosalia Laza Fidalgo,, ―Improving the Revision Stage of a CBR System with Belief Revision Techniques‖, Computing and Information Systems, Vol. 8, p.40-45, 2001

[8]. Cherkassy B V, Goldberg A V and Radzik T. (1993) Shortest Paths Algorithms:Theory and Experimental Evaluation. Research project, Department of Computer Science, Cornell and Stanford Universities and Krasikova Institute for Economics and Mathematics.

[9]. Saunders, S., and Takaoka, T. Improved shortest path algorithmsfor nearly acyclic graphs. In Proc. Computing: The Australasian TheorySymposium, vol. 42 of Electronic Notes in Theoretical Computer Science. 2001.

[10]. V. Batz, D. Delling, P. Sanders, and C. Vetter. Time-Dependent Contraction Hierarchies. In Proceedings of the 11th Workshop on Algorithm Engineering and Experiments (ALENEX'09). SIAM, 2009.

[11]. R. Bauer and D. Delling. SHARC: Fast and Robust Unidirectional Routing. In I. Munro and D. Wagner, editors, Proceedings of the 10th Workshop on Algorithm Engineering and Experiments (ALENEX'08), pages 13–26. SIAM, 2008.

[12]. R. Bauer, D. Delling, P. Sanders, D. Schieferdecker, D. Schultes, and D. Wagner. Combining Hierarchical and Goal-Directed Speed-Up Techniques for Dijkstra's Algorithm. In C. C. McGeoch, editor, Proceedings of the 7th Workshop on Experimental Algorithms (WEA'08), volume 5038 of Lecture Notes in Computer Science, pages 303–318. Springer, June 2008.

[13]. R. Bauer, D. Delling, and D. Wagner. Experimental Study on Speed-Up Techniques for Timetable Information Systems. In C. Liebchen, R. K. Ahuja, and J. A. Mesa, editors, Proceedings of the 7th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems (ATMOS'07), pages 209–225. Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany,2007.

[14]. T. M. Chan, A. Efrat, and S. Har-Peled. Fly Cheaply: On the Minimum Fuel Consumption Problem. Journal of Algorithms, 41(2):330–337, 2001.

[15]. G. B. Dantzig. Linear Programming and Extensions. Princeton University Press, 1962.

[16]. D M Segura Velandia and A West, "Image-based retrieval in case-based reasoning systems for polyurethane manufacture", Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture 2009 223: 89.

[17]. Edmund K. Burke, Bart L. MacCarthy, Sanja Petrovic1, Rong Qu, "Multiple-Retrieval Case-Based Reasoning for Course Timetabling Problems", Journal of Operations Research Society, Vol. 57 Issue 2, pp. 148-162, 2006.

[18]. Ya-jun Jiang, Jun Chen, Xue-yu Ruan, "Fuzzy similarity-based rough set method for case-based reasoning and its application in tool selection", International Journal of Machine Tools & Manufacture 46 (2006) pp. 107–113.

[19]. Mingyang Gu, Agnar Aamodt and Xin Tong, "Component Retrieval Using Conversational Case-Based Reasoning", Intelligent Processing II, 2004, pp. 189-196.

[20].

