An Efficient A^* based Algorithm for Optimal Graph Matching applied to Computer Vision

Douglas Antony Louis Piriyakumar and Paul Levi
Institute for Parallel and Distributed High-Performance Systems
Department of Computer Science, University of Stuttgart
D-70565 Stuttgart, Germany
Phone: +49-711-7816-358 or 387, Fax: +49-711-7816-250
E-Mail: $\{$piriyaku$|$levi$\}$@informatik.uni-stuttgart.de

Abstract.

Many of the problems in artificial intelligence including computer vision demand expensive and computationally intensive searches. Particularly in computer vision, the crux of the problem is usually to match two abstract representation, mostly graphs. Eventhough several approaches are used, to ascertain optimality the A^* strategy is chosen. In this paper, we have presented an efficient A^* based algorithm for optimal graph matching with the incorporation of two techniques additionally. The lower and upper bound techniques help to reduce the number of nodes generated to the appreciable amount and thereby the computation time. The heuristics used in this algorithms provides further reductions asserting the proper choice.

Keywords: Computer vision, Graph matching, Optimal matching, Heuristic methods, A^* Algorithm

1 Introduction

In most of the core application problems viz. artificial intelligence, code optimization in compilers, CAD and computer vision, manouvring the combinatorial search remains still to be solved efficiently. Especially in computer vision, the crux of the problem is to match two abstract representations (Graphs) [SlHr 81].

As early as in 1964 [StHU 64], a heuristic program for testing pairs of directed line graphs for isomorphism was designed. Using representative graphs and reordered graph, another efficient algorithm for graph isomorphism was presented in [CDGC 70]. With backtrack procedure, directed graph isomorphism was solved in [BeAT 73]. Following this, a fast backtracking algorithm for the same not necessarily running in polynomial time was developed [ScDC 76]. An algorithm for subgraph isomorphism using graph theoretical methods is presented in [UlJR 76].

Mostly, two approaches viz. state-space method with branch and bound techniques [LeWd 96] and nonlinear optimization methods with heuristic approximations [CKPm 95] are employed to match graphs efficiently. Recently, noise included graph matching [GsRa 96] and parallel algorithms [ACTS 97] are also investigated. Various strategies and applicability of graph matching to computer vision is explained in [BDBC 82].

However, here we combine the two methods to efficiently get the optimal matching always. The optimality is guaranteed by using A^* algorithm [PMPL 98] with the proper h^* function aptly suiting to this problem. This demands the formulation of the problem in terms of A^* approach and developing heuristics for supplanting the upper bound for matching. The results of optimality have been verified by the enumeration of permutations method.

2 The Matching Problem

2.1 The Definition

Given two graphs, G_1 and G_2 with vertices sets V_1 and V_2 along with edges sets E_1 and E_2, we consider here the number of vertices in both the graphs to be same(say n). A cost matrix C is defined with c_{ij} as the cost involved in matching v_i of G_1 and v_j of G_2. There are several issues taken into consideration for incorporating them in the process of matching viz. degree of mismatch [ACTS 97] and such others like difference between indegree and outdegree. The problem is to find a matching vector M where m_i is the vertex in G_2 matched with the vertex v_i in G_1 such that

$\Sigma c_{i,m_i}$ is minimal $\forall$ i ranging from 1..n.

2.2 The Formulation

Each child node in the state-space of A^* (explained in the next section) denotes a partial assignment i.e., assigning a non-assigned vertex in G_1 with a non-assigned vertex G_2 apart from the already available such assignments made in the parent node. Here, $f^*(x) = g^*(x) + h^*(x)$ where f^* is the cost of the node, g^* is the cost of getting this node from the start node i.e., the cost of the parent node and h^* is the lower bound on the cost of arriving at a solution node from this node i.e., the sum of the static levels of the non-assigned vertices in G_1. The rest is the same as the general A^* strategy [PMPL 98].

3 The New A^* Based Algorithm for Graph Matching

3.1 General A^* Algorithm

As our algorithm is based on the A^* algorithm, for the sake of clarity and explaining our algorithm, the general A^* algorithm used in most of the artificial intelligence problems is given here as in [NiNJ 80]. In A^* algorithm, the state space graph is a tree called search tree. Each node in the tree corresponds to the assignment of a particular vertex in a graph with a specific vertex. All the internal nodes in the tree correspond to partial (or incomplete) matching and all external (leaf) nodes in the tree, correspond to either pruned node or complete graph matching. Our problem here is to find the goal node, a leaf node corresponding to the optimal matching. Associated with a node v in the search tree is a cost function f(v) = g(v) + h(v), which is an underestimate for the minimum cost of an assignment, given that it includes the partial matching. The function g(v) is the cost of the path from the root to v and the function h(v) is a lower bound estimation of the minimum cost function h^*(v), from the node v to a leaf node which corresponds to an optimal matching in the subtree rooted at node v.

3.2 The Heuristics Solution

To set the upper bound so that any node with the cost of partial matching or together with h^* also can be pruned, an effective heuristic is defined here. We define a priority list based on the following partial order,
v_i has more priority than v_j provided $\Sigma c_{ik}, \forall k$ is not less than $\Sigma c_{jk}, \forall k$.
The heuristic chooses each time, a vertex v_p from set of all non-assigned vertices such that no other vertex is having a higher priority. And, this vertex is assigned with a vertex v_q in the set of all non-assigned vertices in G_2 such that c_{pq} is minimal considering all such non-assigned vertices. We have also tried separately with N-Queen problems solutions as heuristics.

3.3 The Heuristics Function

The heuristic function define here as follows. At node x, let there be $V_1^{'}$ vertices already assigned. Then,

$g(x) = \Sigma c_{i,m_i}, \forall$ i $\in V_1^{'}$.

Now, to find the f(x) value, we need h(x), heuristic function. To produce always optimal solution, indeed $h^*(x)$ is required. The $h^*(x)$ is defined as,

$h^*(x) = \Sigma c_{i,i'}, \forall$ i $\in V_1 - V_1^{'}$,

where $c_{i,i'}$ be the minimum in the row i. In fact, it is easy to verify that $h^*(x) < h(x)$ to ascertain the optimality.

3.4 New Techniques for Reducing Space and Time

3.4.1 Lower Bound

The lower bound is for the solution which is the minimum possible attainable solution. In the A^* algorithm, the algorithm has to continue even after finding a solution as it need not necessarily be optimal. Now the question lies how can it be proved that the given solution is the optimal solution so that the algorithm can be stopped atonce. The only possible way is that when the given solution is equal to the lower bound solution, obviously there could not be a better solution. Hence, the algorithm can stop. Now, the problem boils down to finding the lower bound solution to the problem which is normally difficult in the general case, is not so in graph matching.

Let $c_{i,i'}$ be the minimum in the row i. Then, the lower bound is defined as the sum of all such row minima. i.e.,

$LB = \Sigma c_{i,i'}, \forall i = 1, n.$

The major problem with lower bound is that the possibility of many to many mapping is probable. However, in case of multiple similar objects, denoted as several occurrences of the same subgraph, this will indeed be more desirable. One should be always careful that all feasible optimal solutions need not necessarily be lower bound solutions. The main advantage is that if the given problem has the lower bound solution, the algorithm terminates atonce it finds such solution, thereby reducing both the memory space required by the further expansions and the time to compute the same.

3.4.2 Upper Bound

The upper bound is a solution which the already available minimum solution. In the A^* algorithm, the algorithm has to evaluate the function f(x) at every node. Supposing that f(x) is greater than upper bound, that node need not to be expanded further. This will not affect the optimality as anyhow by expanding this node, the solution obtained will be more than that of the already available solution.

However it start with, one should have a heuristic solution. So, this algorithm obviously need a heuristic method to solve the problem. This also helps in another way drastically. Supposing that the heuristic solution is equal to the lower bound solution, then the algorithm stops without creating even a single node in using the general A^* algorithm. Even otherwise, this heuristic solution found initially will serve as the upper bound. So, using upper bound, the number of nodes generated are minimized thereby reducing the memory space and CPU time.

3.5 Our Algorithm for Optimal Graph Matching

1. Compute the lower bound solution, LB.

2. Find a heuristic solution, UB using say N-queen problem.

3. IF $(UB! = LB)$ THEN

4. Construct the priority list of vertices.
5. c = 0 (* node count *).
6. Build the initial node N_0 and insert it in the list with f(N_0) = 0.
7. REPEAT
8. Select the node N_k with smallest f value.
9. IF (N_k is not a solution) THEN
    (a) Generate the successors i.e., trying with all unassigned vertices.
    (b) Do the following for each such vertices
    compare the vertex with all other vertices and assign.
    (c) FOR each such assignment N_i DO
        - Check whether it is already there in the list to eliminate the duplication
        - IF (already available) THEN
          Don't add the node
          ELSE
          Compute f(N_i) = g(N_i) + h(N_i) for this node N_i.
          IF (f(N_i) $<$ UB)
          c = c + 1
          Insert it in the list
          IF (N_i is a solution) THEN
          IF (f(N_i) = LB) THEN
          Print the solution and quit.
          IF (f(N_i) $<$ UB) THEN
          UB = f(N_i).
          ENDIF
          ENDIF
          ENDIF
          ELSE
          Prune the node N_i
          ENDIF
          ENDIF

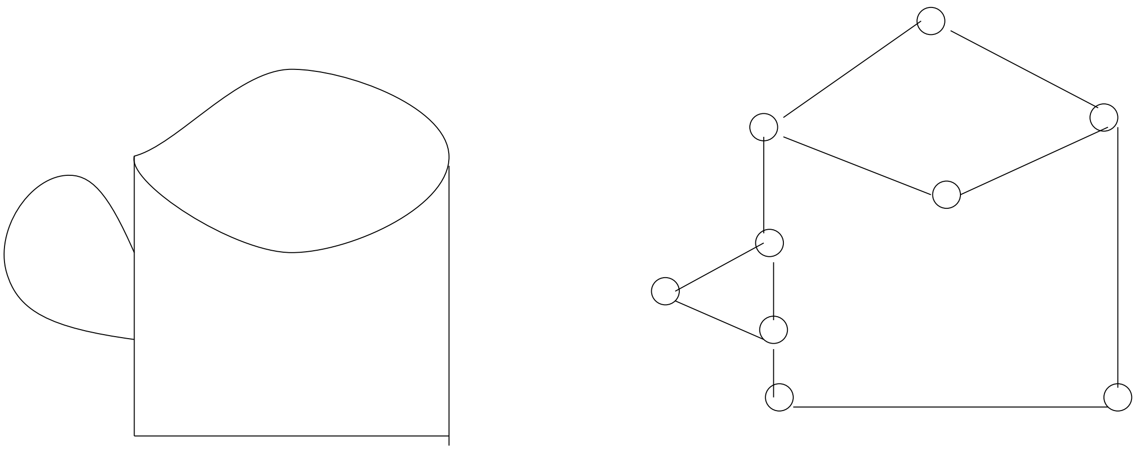
    ENDIF
    ELSE
    Print the solution and quit
10. UNTIL (N_k is solution OR list is empty).

3.6 The Algorithms Developed with Variations

We have developed four variations of the A^* algorithm with our new techniques [PMPL 98] of lower bound and upper bound. Variation A is a simple A^* without employing any technique. Variation B is a simple A^* with the above techniques. Variation C is at each level of state-space tree only one vertex is selected based on the priority list. Variation D is the same as variation C together with these techniques.

TABLE I
Comparison of the variations of A^* and permutation Algorithms

Algorithm	*No. of nodes generated*	*CPU time in sec*
Variation A	285	0.183
Variation B	0	0.083
Variation C	45	0.083
Variation D	0	0.067
Permutation	362880	18.133



(a) (b)

Figure 1: Recognition of Objects. (a) a real life Object (b) a corresponding Graph

4 Result and Analysis

The Table I is for the example given in Fig.1. It is very evident that our algorithms at any case outperform the general permutation method which also assures the optimality.

The variation A is just the A^* algorithm. By adding these techniques to the variation B, the reduction in both memory space and CPU time is evident. The prime difference between the set A,B and C,D is that in step 9, only one vertex is selected each time considering all unassigned vertices depending upon the priority. It may be also recalled by this, the optimality of the algorithm is not sacrificed. By allowing that only there will be lot of duplications which are also properly handled. The variation D is always better than Variation B, eventhough both produces the lower bound solution due to the power of the effective heuristic defined whenever possible. We would like to also inform that these variations are only for those who long for the optimality. The heuristic defined here in this paper as well as the N-Queen problem will also serve the purpose of those who are not interested in optimal solution, but a quick reasonable sub-optimal solution. As these algorithms are highly parallelizable, we are proceeding now with parallelization. We have tried to do permutation of the input cost matrix. However, the results are almost the same for the cases we have tried. To check, the number of nodes were reduced from 9 to 7 and the whole procedure is repeated. Table II portrays the same vividly.

TABLE II
Comparison of the variations of A^* and permutation Algorithms for lesser number of nodes

Algorithm	*No. of nodes generated*	*CPU time in sec*
Variation A	140	0.1
Variation B	0	0.083
Variation C	28	0.083
Variation D	0	0.067
Permutation	5040	8.7

5 Conclusion

The problems in artificial intelligence including computer vision which demand expensive and computationally intensive searches are tried for optimal solution. Eventhough several strategies are applied to solve the matching of abstract representation, mostly graphs, we have chosen A^* algorithm to ascertain optimality. The efficient A^* based algorithm for optimal graph matching with the incorporation of two techniques additionally is presented in this paper. The lower and upper bound techniques help to reduce the number of nodes generated and the execution time to the appreciable amount. The heuristics used in this algorithms provides further reductions asserting the proper choice. The role of initial heuristic solution, in both upper and lower bound techniques are also exemplary and well suited to the graph matching problem. The variants of the algorithm presented here strengthen the claims of optimality clearly.

References

[ACTS 97] Allen, R., Cinque, L., Tanimoto, S., Shapiro, L. and Yasuda, D., "A Parallel Algorithm for Graph Matching and its Maspar Implementation," IEEE Trans. Parallel and Distributed Systems, vol. 8, no. 5, pp. 490-500, May 1997.

[BDBC 82] Ballard, D.H. and Brown. C.M., *Computer Vision*, Englewood Cliffs, NJ, Prentice-Hall, 1982.

[BeAT 73] Berztiss, A. T., ”A Backtrack Procedure for Isomorphism of Directed Graphs,” Journal of Association of Computing Machinery, vol. 20, no. 3, pp. 365-377, July 1973.

[CKPm 95] Christmas, W.J., Kittler, J. and Petrou, M., ”Structural Matching in Computer Vision using Probabilistic Relaxations,” IEEE Trans. PAMI, vol 17, no. 8, pp. 749-764, Aug. 1995.

[CDGC 70] Corneil, D.G. and Gotleib, C.C., ”An Efficient Algorithm for Graph Isomorphism,” Journal of the Association for Computing Machinery, vol. 17, no. 1, pp. 51-64, January 1970.

[GsRa 96] Gold, S. and Rangarajan, A., ”A Graduated Assignment Algorithm for Graph Matching” IEEE Trans. PAMI, vol 18, no 4, pp.377-388, April. 1996.

[LeWd 96] Lawler, E. and Wood, D., ”Branch and Bound Methods: A Survey,” Operations Research, vol 14, pp. 699-719, July-Aug. 1966.

[NiNJ 80] Nilson, N.J. *Principles of Artificial Intelligence*, Palo Alto, Calif, Tiaga Publications, 1980.

[PMPL 98] Piriyakumar, D.A.L., Murthy, C.S.R. and Levi, P. ”A new A^* Based Optimal Task Scheduling in Heterogeneous Multiprocessor Systems Applied to Computer Vision,” HPCN' 98 International conference, Amsterdam, April 21-23, 1998.

[ScDC 76] Schmidt, D.C., ”A Fast Backtracking Algorithm to Test Directed Graphs for Isomorphism Using Distance Matrices,” Journal of Association of Computing Machinery, vol. 23, no. 3, pp. 433-445, July 1976.

[SlHr 81] Shapiro, L.G. and Haralick, R.M., ”Structural Descriptions and Inexact matching,” IEEE Trans. PAMI, vol 3, pp. 504-519, sept. 1981.

[UlJR 76] Ullmann, J.R., ”An Algorithm for Subgraph Isomorphism,” Journal of Association for Computing Machinery, vol. 23, no. 1, pp. 31-42, January 1976.

[StHU 64] Unger, S.H., ”GIT - A Heuristic Program for Testing Pairs of Directed Line Graphs for Isomorphism,” Communications of the ACM, vol. 7, no. 1, pp. 26-34, January 1964.