Fast Computation of the Shortest Path Problem through Simultaneous Forward and Backward Systolic Dynamic Programming

Nuha A. S. Alwan
Department of Computer Engineering
College of Engineering
University of Baghdad, Iraq

Ibraheem K. Ibraheem
Department of Computer Engineering
College of Engineering
University of Baghdad, Iraq

Sabreen M. Shukr
Department of Computer Engineering
College of Engineering
University of Baghad, Iraq

ABSTRACT

A systolic parallel system based on simultaneous forward and backward dynamic programming is proposed for the solution of the shortest path problem. The speed-up advantage of this fast systolic solution to this problem is very important in applications such as shortest-path routing in wireless networks for instance. The merit of this method becomes clear in a straightforward manner when the number of stages of the directed acyclic graph (DAG) through which the shortest path is derived is odd, although an even number of stages can also be accounted for.

General Terms

Systolic Parallel Processing, Shortest Path, Algorithms, Dynamic Programming.

Keywords

Systolic array; directed acyclic graph; dynamic programming; least cost; latency; throughput.

1. INTRODUCTION

A graph is a set of vertices and a collection of edges that each connects a pair of vertices. The most intuitive graph-processing problem is the shortest-path problem which is concerned with finding the lowest-cost way to get from one vertex to another [1]. The graphs under consideration are edge-weighted and have an associated path weight, the value of which is the sum of the weights of the path's edges. A prominent application of the shortest-path problem is routing in multi-hop wireless networks which employs shortest-path and graph theory-based algorithms such as dynamic programming (DP) algorithms. Every wireless channel between two nodes (vertices) is viewed as a link (edge). In dynamic (adaptive) routing, the nodes need to be notified to recalculate their routes in prompt response to topology changes. This calls for intensive parallel processing in the nodes [2] to prevent bottlenecks which counteract the high speed merits of wireless networks. The ever-increasing capacity of wireless transmission [3] necessitates node switching systems with capacity and rapid response to meet this speed demand. In this paper, we attempt to calculate the best path in each vertex by employing systolic parallel dynamic programming assuming general directed acyclic graphs (DAGs). Since a DAG is a directed graph with no directed cycle, all edges have identical directions to a destination-oriented DAG root. DAGs differ from trees in that a child vertex can have more than one parent vertex [4].

The special and distinguishing feature of a DAG is that its vertices can be linearized [5, 6], i.e. they can be arranged on a line so that all edges go from left to right for instance as in Figure 1. More specifically, DAGs can always be linearized into stages [6,7] allowing us to derive algorithms suitable for multistage graphs [7]. This may be achieved via dynamic programming, which is a technique of very broad applicability that can be imported when more specialized methods fail. Dynamic programming is a powerful algorithmic paradigm in which a problem is solved by identifying a collection of sub-problems and tackling them one by one, smallest first, using the answers to small problems to help figure out larger ones, until the whole lot of them is solved [5,8].

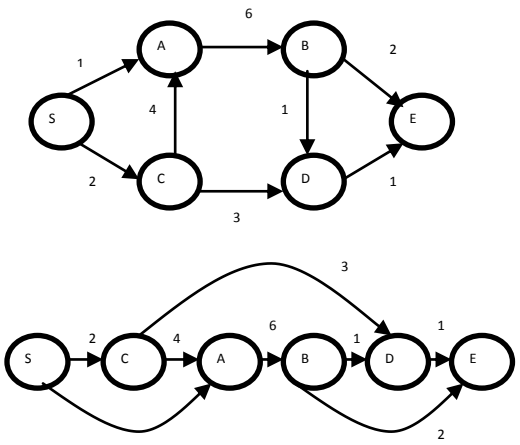

Fig 1: Linearization of DAGs

Dynamic programming has been proven to be effective for many single- and multi-objective optimization problems. For some problems, it is even the most efficient approach known for the solution. There are numerous other optimization methods each with advantages and disadvantages. The most prominent is dynamic programming due to its generality, reduction of complexity and facilitation of numerical computation, simplicity in incorporating constraints, and its conformity to the stochastic nature of some problems [9,10,11]. Evolutionary algorithms such as genetic algorithms (GA) are most appropriate for complex non-linear models where the location of a global minimum is a difficult task. Due to global search, GAs are computationally expensive and

need more time to be computed as compared with dynamic programming [9].

In this work, we consider forward and backward dynamic programming as shortest-path finding algorithms in a multistage graph. These techniques readily lend themselves to systolic implementation. Systolic architectures [12] are parallel processing architectures with local interconnections between modular processing elements (PEs) with regular identical functions. These are very desirable features for very large scale integration (VLSI). In a systolic array, data paths are ideally fully pipelined. The parallelism results in a computational speed advantage.

The rest of the paper is organized as follows: Section 2 discusses forward and backward dynamic programming for directed graphs and the related issues of how DAGs can be accounted for within this framework. Section 3 describes the systolic DP implementation. Section 4 improves the work presented in Section 3 to further increase the speed-up advantage by using simultaneous forward and backward systolic dynamic programing resulting in a "fast" computation for the problem at hand. Section 5 concludes the paper.

2. DYNAMIC PROGRAMMING ALGORTHMS FOR DAGs

Dynamic programming aims at creating a sequence of interrelated decisions in an optimum way. The key attribute that a problem must have in order for dynamic programming to be applicable is optimal substructure [13]. Optimal substructure means that the solution to a given optimization problem can be obtained by the combination of optimal solutions to its subproblems. Consequently, the first step towards devising a dynamic programming solution is to check whether the problem possesses such optimal substructure. For a given weighted graph G=(V,E) where V is the set of vertices and E the edges, the shortest path or least-cost path p from a vertex u to vertex v exhibits optimal substructure: taking any intermediate vertex w on this shortest path p, if p is truly the shortest path, then path p1 from u to w and path p2 from w to v are indeed the shortest path between the corresponding vertices. Hence one can easily formulate the solution for finding shortest paths in a recursive manner [5].

In [8], it is shown that there are two fundamental processes to solve the shortest path problems in multi-stage directed graphs. These are very suitable to systolic implementation as will be described shortly in Section 3. The first one is known as forward dynamic programming which works as follows: after converting a given DAG network representation to an n-stage graph G(V, E) as shown in Figure 2 in which n=4, we can find the shortest path by starting at a destination vertex d and working to reach an initial vertex s. (The explanatory vertex and stage symbols discussed here are not to be confused with the cost or weight symbols in Figure 2).

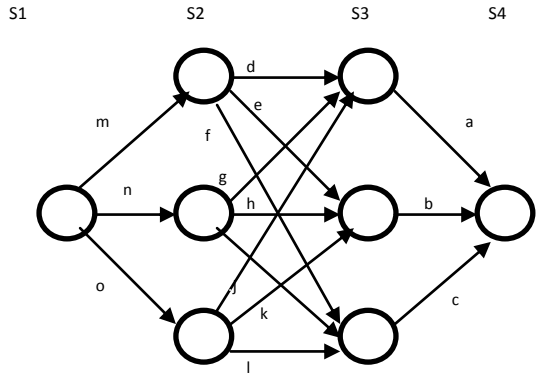

Fig 2: A multi-stage (four-stage) directed graph with weights shown on edges. The weights are denoted by letters. Enumeration or labeling of edges is not shown.

The function F(i, j) is defined to be the cost of the path to go from the vertex number j at stage i to the destination vertex. First, we enumerate all possible edges and keep track of the shortest path enumeration along the dynamic programming procedure. Beginning at the last stage, we can determine the F(n-1, j) without any computation. Then we determine the F(n-2, j) by adding the costs of the two edges that connect vertex j at stage n-2 to vertex d. Next, we choose the path with minimum cost among all paths from that vertex j at stage n-2 to vertex d, and assign that path as the shortest path from vertex j at stage n-2 to the destination vertex d. Afterwards, all the shortest paths from all vertices in all stages to the vertex d must be determined in the same way in order to find the shortest path from vertex s to vertex d.

The other process is known as backward dynamic programming which starts from an initial vertex s and works toward a destination vertex d. The function B(i, j) is defined to be the cost of the path to go from vertex s to the vertex j at stage i. We start from the second stage, and determine the B(2, j) without any computations. After that, we determine the B(3, j) by adding the costs of the two edges that connect the vertex s to vertex j at stage 3. Next, we choose the path with the minimum cost among all paths that connect vertex s to vertex j at stage 3, and choose that path as the shortest path from vertex s to vertex j at stage 3. All shortest paths from vertex s to all vertices in all stages must be determined in the same manner in order to find the shortest path from vertex s to vertex d.

It has been explained and shown in Figure 1 that DAGs can be linearized into stages. This is true albeit the fact that edges skipping stages must always be avoided. Stage skipping can be made up for in forward and backward dynamic programming by assuming that a skipping edge can be connected, as well, to one or more intermediate vertices between the edge's local source and destination vertices. This assumption is validated if all the edges representing a skipping edge are given the same weight and the same label or enumerating number. Although the resulting multi-stage directed graph does not describe exactly what is actually happening, but our dynamic programming methods can be applied to it giving exactly the same results. When the shortest path is deduced, consecutive edges with the same label are considered as one skipping edge.

3. SYSTOLIC IMPLEMENTATION OF FORWARD AND BACKWARD DYNAMIC PROGRAMMING

A systolic array, based on dynamic programming, which solves the shortest-path optimization problem, is shown in Figure 3. Let us assume a 4-stage directed graph with each stage consisting of three vertices except the stage with source and destination vertices, in accordance with Figure 2. In general, some of the paths may not exist, but we account for them in the systolic algorithm by assuming they are present with infinite cost. In our case, the array will then consist of three processing elements (PEs) or cells.

In reference to Figure 2, the edge costs a, b, and c are preloaded into the array cells. The other edge costs are entered skewed and in the order shown in Figure 3. At time t=0, d will be added to a, and the sum of the two will be considered as the cost of the path d-a. The two labels of this two-edge path are concatenated and stored together with the path cost. This path cost is now compared to the infinite value entered at the bottom-most cell, and the minimum together with the concatenated edge labels is moved to the cell above with the new clock cycle. This last trivial comparison is necessary to maintain the modularity and regularity required by the systolic array. With the advent of the new clock trigger, e is added to b in the second cell and the result is compared with that received from the bottom-most cell. The minimum will move to the topmost cell with the third clock trigger. The edge cost g will have been compared to a at the second clock pulse at the bottommost cell, and so on.

After three clock pulses, the minimum two-edge path cost (termed F(2, 1) in accordance with Section 2) together with the path labels are output from the topmost cell, followed by the minimum of the paths g-a, h-b and i-c (F(2, 2)), and so on. At the sixth tick of the clock, the minimum path costs corresponding to the three vertices of stage S2 are made to reside in the cells, replacing a, b, and c, by means of the unshaded delay elements. At this same instant, computations for the new stage begin.

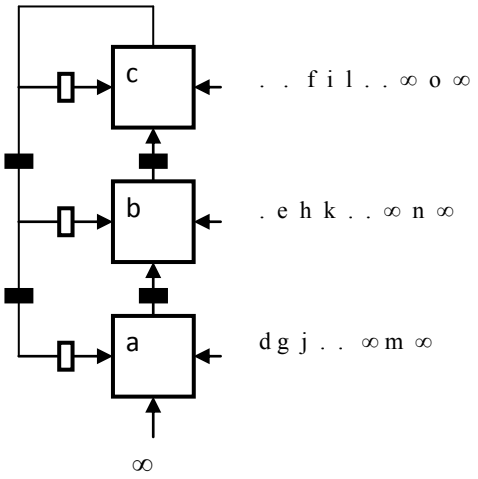

Fig 3: Systolic array implementation of forward dynamic programming for the directed graph of Fig 2.

The trigger to the unshaded delay elements is the output of a divide-by-five counter triggered by the master clock. Naturally, the master clock triggers the shaded delay elements.

Now the array is ready to receive the next-stage (S1) edge costs. These are simply the edge costs from the source vertex to each vertex of stage S2. For generality, S1 is considered as being made up of three vertices also, but the skewed infinite input values shown in Figure 3 account for the actually missing vertices. Likewise, any missing edges in the graph stages are assigned infinite cost.

At the end of the process, precisely at the 9^{th} tick of the clock, the edge labels of the final path with the minimum cost, together with this minimum cost, are stored in the topmost cell. Thus, all cells perform exactly the same operations in a pipelined rhythmic fashion. From Figure 3, we see that this systolic algorithm results in some underutilization of the array cells. This may be noticed from the blank input entries between stage inputs. In-cell registers are used to store the minimum cost and successive edge labels that represent the most recent shortest sub-path.

The following pseudo-code summarizes the proposed systolic algorithm using forward dynamic programming (Figure 3):

```
1.  INPUT: Number of stages n, number of vertices or PEs
          v, cell-resident first edge costs a, b, and c.
2.  OUTPUT: Shortest path and optimized cost.
3.  /*Initialization*/
4.  Cell inputs and outputs=0, bottommost cell input=∞
5.  Delay-element inputs and outputs=0
6.  /*Main loop*/
7.  FOR stage number = n-2 to 1 step -1 DO
8.      FOR time instant = 1 to 2v-1 DO
9.          /* Trigger all shaded delay elements to commu-
            nicate their inputs to their outputs*/
10.         Shaded delay-element output = its input
11.         FOR ALL (cells in (1….v)) DO
12.             Sub-path cost = right-side input+resident
                value
13.             Shortest path = concatenated labels
                of edges involved
14.             IF sub-path cost > lower input THEN
15.                 Optimized sub-path cost of current
                    stage = lower input
16.                 Shortest path = concatenated labels of
                    edges from lower input
17.             ELSE
18.                 Optimized sub-path cost of
                    current stage = sub-path cost
                    /*Shortest path unchanged*/
19.             ENDIF
20.         END FOR
21.     END FOR
22.     /*Trigger all unshaded delay elements to communi-
          cate their inputs to their outputs to replace cell-
          resident values*/
23.     Unshaded delay-element output = its input
24. END FOR
25. RETURN Optimized sub-path cost
26. RETURN Shortest path
```

There is yet no standard pseudo-code for parallel algorithms. However, recently [14], a language has been developed for scalable parallel computers. Thus, in line 11, the keyword "FORALL" and its statement have been borrowed from [14] to indicate operations executed in parallel threads. That is, the three PEs operate simultaneously and systolically, performing identical operations at each time instant. The infinite inputs are actually suitably large numbers determined by the foreseeable costs for the specific problem at hand, or could be the largest number that the word length in use can accommodate

.A systolic array that achieves backward dynamic programming for the same DAG of Figure 2 is shown in Figure 4. Notice the difference in entering the edge costs in Figure 4 as compared to Figure 3.

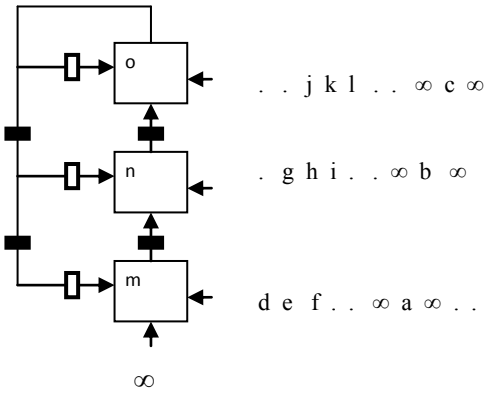

Fig 4: Systolic array implementation of backward dynamic programming for the directed graph of Fig 2.

In Figure 3, the operations of the constituent cell are one addition and one comparison operation (subtraction and sign bit control). Therefore, the throughput is equal to [6fc.(3/5)] addition/subtractions per second, where fc is the master clock frequency since there are three PEs each with two successive operations. The (3/5) factor is included to account for the PE underutilization present. To speed up the process, the clock period Tc=1/fc can be made as small as the time required by two addition-subtractions only. The latency is nine clock periods. This is the time needed until the path with the minimum cost is output from the topmost cell together with the value of this path cost. This is obvious form Figure 3 if we count the number of inputs (including blank inputs) to the topmost cell until all but the last edge costs have been entered.

Several systolic array implementations of the dynamic programming problem have been proposed in the literature such as in [15,16]. In [15], the multi-stage graph problem with a forward dynamic programming formulation is solved in analogy to multiplication of a string of matrices. However, the array is only semi-systolic as it employs global interconnections and is not fully pipelined. A systolic array for single-source shortest-path problems is derived in [16] systematically as opposed to our ad hoc design. A cubic dependence graph is obtained and projected onto a two-dimensional systolic array. The spatial complexity of this array is further reduced by multiprojection onto a one-dimensional systolic array that is comprised of N PEs, where N is the total number of vertices in the multi-stage graph. This is also equal to the data pipeline period of the array. N periods are further needed to produce array outcomes resulting in a latency of $O(N^2)$ clock periods. In contrast, the number of PEs in the present work is equal to the maximum number (M) of vertices per stage. Naturally, M is smaller than N. Thus, the latency of our array is of O(M) clock periods. The merit of the method in [16], however, is that the systematically designed systolic array has a generality that makes it applicable to other dynamic programming problems.

4. THE IMPROVED (FAST) SYSTEM

To further increase its computational speed, the systolic architecture considered in this work can be modified by employing both forward and backward dynamic programming with two systolic arrays simultaneously when the number of stages is odd. Each systolic array will compute half the shortest path such that the overall throughput is doubled and the latency halved. If the number of stages is even, and to maintain the symmetry of the systolic architecture, a virtual stage can be added, the vertices of which are skipped by the edges of the preceding stage to the stage following the virtual stage. This skipping is then accounted for as explained at the end of Section 2.

The operation of the proposed system is clarified by the following example with the aid of Figures 5 through 7. Figures 5 and 6 demonstrate a seven-stage DAG with weighted edges. The middle stage is S4 which is shown in both figures. The three shortest paths from S1 (single vertex) in Figure 5 to each vertex of S4 can be found systolically and consecutively employing backward DP as explained in Section 3. Likewise, the shortest paths from each vertex in S4 to S7 (single vertex) is found systolically and consecutively employing forward DP. The respective right-hand and left-hand systolic arrays in Figure 7 achieve this end. Figure 7 is the overall systolic architecture of the proposed fast system. The systolic arrays in Figure 7 operate simultaneously when the edge weights are entered to them skewed in time and in the order explained in Section 3.

At the eighth clock pulse, the shortest paths from S1 to the first vertex in S4, and from this latter vertex to S7 will be available at the topmost cells of the right-hand and left-hand systolic arrays respectively. The gray-shaded delay elements are triggered by a divide-by-8 counter output. Thus, at the ninth clock pulse, the two afore-mentioned paths will be input each to one side of the middle cell shown in Figure 7. Their costs will be summed in this cell, and their edge labels concatenated. This minimum cost is compared to infinity for regularity and stored together with the path labels. At the tenth clock pulse the two-part shortest path corresponding to the second vertex of S4 will be available at the middle cell. They will likewise be cost-summed and label-concatenated, and compared to the result of the previous time instant. The shortest path with the minimum cost will be stored. Therefore, we have the overall shortest path and its cost available at the output of the middle cell at the eleventh time instant.

If this problem were solved using only one systolic array employing forward or backward DP, the final result would be available at the twentieth clock cycle. This means that the latency is almost halved with the fast technique. Clearly, the throughput is almost doubled since the two arrays operate simultaneously each with almost or exactly half the number of stages (depending on whether the number of stages is odd or even respectively).

5. CONCLUSIONS

A systolic algorithm based on forward dynamic programming has been proposed for the solution of the shortest path problem. The throughput and latency of the proposed systolic router are [6fc.(3/5)] and [9/fc] respectively, where fc is the master clock frequency, when the number of vertices per stage is equal to three. Comparing with existing dynamic programming systolic arrays, ours is an improvement in terms of latency, though throughput is comparable. Moreover, our algorithm is fully pipelined resulting in a highly systolic structure. Our systolic architecture can make up for DAG-based network structures by converting them into multi-stage directed graphs through repeated link labeling. The systolic architecture has been modified to increase its computational speed by using both forward and backward dynamic programming simultaneously to advantage to obtain doubled throughput and halved latency.

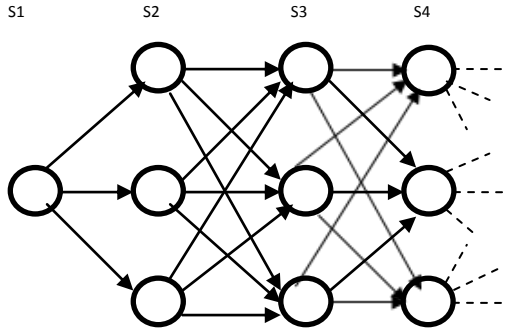

Fig 5: Stages S1 to S4 of the 7-stage directed graph. Shortest path found by left-hand systolic array of Fig 7.

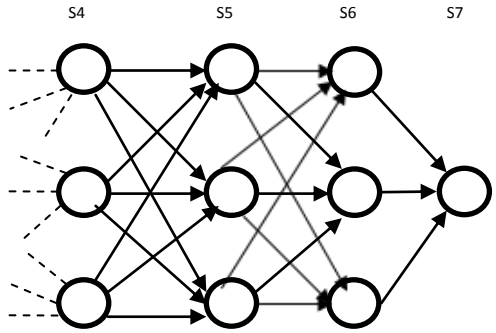

Fig 6: Stages S4 to S7 of the 7-stage directed graph. Shortest path found by right-hand systolic array of Fig 7.

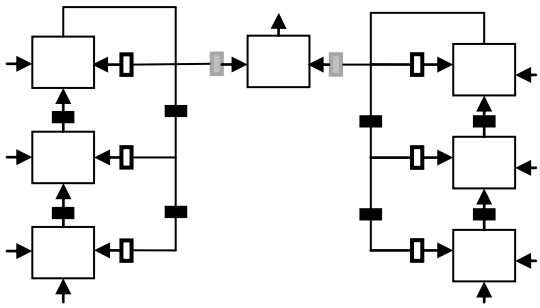

Fig 7: The systolic architecture of the improved (fast) system

6. REFERENCES

[1] Sedgewick, R., Wayne, K. 2011. Algorithms. Fourth edition, Addison-Wesley, Upper Saddle River, NJ.

[2] Stallings, W. 2011. Data and Computer Communications. Ninth edition, Pearson.

[3] Glisic, S., Lorenzo, B. 2009. Advanced Wireless Networks. Second edition, Wiley.

[4] Hong, K. S., Choi, L. 2010. DAG-Based Multipath Routing for Mobile Sensor Networks. Online: http://it.korea.ac.kr /research/publication/papers/DMR-ICTC2011.pdf.

[5] Dasgupta, S., Papadimitriou, C., Vazirani, U. 2007. Algorithms. McGraw-Hill.

[6] Gansner, E. R., Koutsofios, E., North, S. C., Vo, K. P. 1993. A technique for drawing directed graphs. IEEE Transactions on Software Engineering, vol. 19, no. 3, March, 1993. 214-230.

[7] Elmallah, E. S., Culberson, J. 1995. Multicommodity Flows in Simple Multistage Networks. Networks, vol. 25. 19-30.

[8] Moon, T. K., Stirling, W. C. 2000. Mathematical Methods and Algorithms for Signal Processing. Upper Saddle River, New Jersey: Prentice-Hall.

[9] Doerr, B. Eremeev, A., Horoba, C., Neumann, F., Theili, M. 2009. Evolutionary Algorithms and Dynamic Programming. GECCO'09: Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation, ACM, New York, NY, USA. 771-778.

[10] Lew, A., Mauch, H. 2007. Dynamic Programming: A Computational Tool. Springer-Verlag, Berlin Heidelberg.

[11] Ilaboya, I. R., Atikpo, E., Ekoh, G. O., Ezugwu, M. O., Umokoro, L. 2011. Application of Dynamic Programming to Solving Reservoir Operational Problems. Journal of Applied Technology in Environmental Sanitation, vol. 1, no. 3 (Oct. 2011). 251-262.

[12] Petkov, N. 1993. Systolic Parallel Processing. Elsevier Sci. Publ., North-Holland.

[13] Carmen, T. H., Leiserson, V. E., Rivest, R. L., Stein, C. 2001. Introduction to Algorithms. Second Edition, MIT.

[14] Tang, P. 2008. Extending Parallel Pseudo-Code Language Peril-L. PDCCS'08: Proceedings of the 21st International Conference on Parallel and Distributed Computing and Communication Systems, New Orleans, LA, USA. 38-43, September 2008.

[15] Li, G. J., Wah, B. W. 1985. Systolic processing for dynamic programming problems. Proceddings of the International conference on parallel processing, University Park, PA, UST. 434-441, August 1985.

[16] Lee, J. J., Song, G. Y. 2002. Implementation of the systolic array for dynamic programing. Proceedings of the International conference on information technology and applications, ICITA, Bathurst, Australia. 24-28, November 2002.