
A Genetic Algorithm Applied to the Maximum Flow Problem

Toshinori Munakata

Computer and Information Sciences Department
Cleveland State University
Cleveland, OH 44115
email: munakata@cis.csuohio.edu

David J. Hashier

Eaton
1111 Superior Ave.
Cleveland, OH 44114

Abstract

A genetic algorithms is applied to find a maximum flow from the source to sink in a weighted directed graph, where the weight associated with each edge represents its flow capacity in one direction. The maximum flow problem appears to be more challenging in applying genetic algorithms than many other common graph problems because of its several unique characteristics. For example, a flow at each edge can be anywhere between zero and its flow capacity, rather than, say, a fixed distance. Also, the total inflow and outflow at each vertex must balance. In our approach, each solution is represented by a flow matrix. The fitness function is defined to reflect two characteristics - balancing vertices and the saturation rate of the flow. Starting with a population of randomized solutions, better and better solutions are sought through the genetic algorithm. Optimal or near optimal solutions are determined with a reasonable number of iterations compared to other previous GA applications.

1. INTRODUCTION

The maximum flow problem is one of several well known basic problems for combinatorial optimization in weighted directed graphs. Because of its importance in many areas of applications such as computer science, engineering and operations research, the maximum flow problem has been extensively studied by many researchers using a variety of methods (Tarjan 1983, McHugh 1990, Mazzoni 1991). They include: a classic approach (Ford 1962), translation of the maximum flow problem into maximal flow problem in layered network (Dinic 1970), an $O(n^2 \log n)$ parallel algorithm (Shiloach/Vishkin 1982), $O(n^2 \log n)$ to $O(n^3)$ distributed algorithms (Segall 1982, Marberg/Gafni

1984, Yeh/Munakata 1986), and recent sequential algorithms of, e.g., $O(n^2 m)$ for general and $O(nm \log n)$ for sparse graphs (Ahuja/Orlin 1989, Alon 1990, Ahuja/Orlin 1991), with n vertices and m edges.

The problem is to determine an optimal solution for a given directed, integer weighted graph (See Figure 1). The weight at each edge represents the flow capacity of the edge. Under these constraints, we want to maximize the total flow from the source (Vertex No. 1) to the sink (Vertex No. n). This is a simpler version of the more general maximum flow problem. There are different ways of generalizing the problem. For example, a flow can be in either direction at each edge with possibly different flow capacities, or a graph can have multiple sinks and sources.

Recently genetic algorithms (GA) have been applied to various optimization problems (Davis 1987, Goldberg 1989, Koza 1992), including graph problems (Michalewicz 1991). The maximum flow problem appears to be more challenging in applying genetic algorithms than many other common graph problems (e.g., shortest path, minimum spanning tree) because of its several unique characteristics. For example, a flow at each edge can be anywhere between zero and its flow capacity, i.e., it has more "freedom" to choose. In many other problems, selecting an edge may mean to simply add a fixed distance.

In this paper, we discuss our experiment on genetic algorithm application to the maximum flow problem. In the maximum flow problem, two conditions must be satisfied: (1) The flow at each edge must be between zero and its flow capacity. (2) At each vertex, the incoming flow and outgoing flow must balance. One of difficulties of applying a genetic algorithm is that crossing over two solutions may result in unbalanced conditions at the crossed over vertices.

2. OVERVIEW

Generating a balanced graph using the standard processes of reproduction, crossover, and mutation proved to be a difficult task. Initial attempts which could solve small simple graphs would fail with the addition of bottlenecks, loops, and vertices to the graph. The primary problem was the inability of a vertex in the graph to adjust to the flow level of the adjacent vertices. Our algorithm uses dominance and a variable mutation rate which allows a vertex to react to adjacent vertices and find near maximum flow solutions for large graphs which include bottlenecks and loops.

This algorithm selects two members from the current population and builds a new graph using the best or dominant vertices from each member. A dominant vertex is determined by a comparison of the energy level of the same vertex from each member. The energy level is determined by the balanced condition and flow capacity through the vertex. A vertex that is balanced (inflow = outflow) and has a flow at maximum capacity has an energy level of 0 and is considered stable. A vertex that is out of balance (inflow \neq outflow) and the flow capacity is less than maximum has an energy level other than zero and is unstable. The dominant vertex is the vertex with the lowest energy level.

The stability of a vertex is related to the energy level. As the energy level rises a vertex becomes increasingly unstable. A vertex always seeks stability and will give up energy (flow) in order to stabilize. The mutation rate applied to each edge in a vertex is related to the balance component of the energy level of the vertex. Edges in a vertex with a high energy level will have a high probability of mutation in the direction which will lower the energy level of the vertex. This operation does not balance the vertex but provides for motion towards a balanced vertex.

When a dominant vertex is assimilated by the new graph the incoming and outgoing edges may be further modified. For example, assume the dominant vertex has an incoming edge with a flow of 10 from vertex A. Assume vertex A has already been added to the new graph and is showing an outgoing edge to the dominant vertex with a flow of 7. In this situation the new edge may be adjusted to a new flow level of 8 (truncate $(7+10)/2$). While this process may adversely effect the current graph and create instabilities in these vertices, future generations have the means to adapt and can use this information to balance the flow through the entire graph.

Before describing the details of the algorithm, we give a note on its complexity. Analytically determining the

complexity of the algorithm is difficult since the number of generations required to produce a good solution is hard to predict. For sequential implementation, the complexity for one generation can be represented as $O(np)$, where p is the population size (or $O(mp)$ if $m \gg n$). The entire complexity for sequential implementation is then $O(np^2g)$, where g is the number of generations; g may be a function of n and possibly p . For parallel implementation, p solutions in each generation can be processed in parallel, eliminating the factor p . The upper bound of the complexity for each generation is $O(n)$ (up to $O(n)$ in step 3 and $O(\log n)$ or $O(\log m)$ in step 4 in the next section), and $O(ng)$ for the entire operation. Based on our limited experimental experience, g appears to be roughly proportional to n^2 . The population size p has an indirect effect on g in our experiment - when the population size is too small, the probability of not getting balanced solutions within the iteration limits increases. Summarizing these, the entire complexity is roughly $O(n^3p)$ for sequential and $O(n^3)$ for parallel. To compete with the traditional algorithms, further improvements appear to be necessary.

3. ALGORITHM

Let a weighted graph have n vertices and m edges. The graph is represented by the flow capacity matrix, $C = [c_{ij}]$, $i, j = 1, n$. Each solution is represented by a flow matrix $F = [f_{ij}]$, $i, j = 1, n$. (Although this matrix form can be converted to a binary string as in many GA applications, the matrix form is easier to manipulate.) We assign initial flow randomly to every edge. We set f_{ij} to satisfy $0 \leq f_{ij} \leq c_{ij}$.

Iteration process

Repeat the following steps until the condition in Step 3 is satisfied.

Step 1. Compute the following for each of n vertices, $i = 1, n$. ($i = 1$ for the source, $i = n$ for the sink)

$$I_i = \sum f_{pi}, \text{ (Total) inflow to vertex } i, \text{ where } \sum \text{ is taken over the incoming edges to vertex } i. \\ I_i \geq 0.$$

$$O_i = \sum f_{iq}, \text{ (Total) outflow from vertex } i, \\ \text{where } \sum \text{ is taken over the outgoing edges from vertex } i. O_i \geq 0.$$

Excess flow at vertex $i = X_i = I_i - O_i$ for $i = 1, n - 1$, and $X_n = I_n + O_1$. When $X_i = 0$, vertex i is said to be locally balanced.

$$\text{Total flow at vertex } i = T_i = I_i + O_i \text{ for } i = 1, n - 1, \text{ and } T_n = I_n.$$

Step 2. Check globally balanced condition, BALANCE.

If every vertex is locally balanced BALANCE = 1. Otherwise BALANCE = 0 and go to step 4.

Step 3. Compute the following for each of vertex i , $i = n - 1$ to 1, by going backward of the flow directions.

$BLK_i = 1$ if vertex i is blocked; $BLK_i = 0$ otherwise.

Vertex i is *blocked* if for all outgoing edges from vertex i to vertices j 's,

- (i) $f_{ij} = c_{ij}$ (i.e., edge ij is saturated), or
- (ii) vertex j is blocked.

The sink is never blocked.

If $BLK_1 = 1$ (i.e., the source is blocked), then STOP. Otherwise, continue.

Step 4. Reproduction

Our fitness function is determined as follows:

$$\text{fitness } (F) = (\text{Number of balanced vertices}) / (n - 2) - |\text{sum of excess flow}| / (\text{sum of } f_{pi} \text{ over all the } m \text{ edges}) + (\text{sum of } f_{pi} \text{ over all the } m \text{ edges}) / (\text{sum of } c_{pi} \text{ over all the } m \text{ edges})$$

The fitness function reflects two characteristics of the goodness of each solution. The first term represents the ratio of the balanced vertices to the total number of vertices ($n - 2$ since the source and sink always balance). The second term penalizes the solution for the degree by which the graph is out of balance. The third term shows the degree of saturation. We note that all the terms are dimensionless. Although these terms can be multiplied by different constant factors before addition, we selected the above form for simplicity.

Two solutions are selected for reproduction at random from the population. The probability of a specific solution being selected is ratio of the solution fitness to the sum of the fitness of the population. The selected pair of solutions undergoes Step 5 Crossover and Step 6 Mutation. This process is repeated for the population size, giving a new generation.

Step 5. Crossover

The crossover process combines the two solutions F_r and F_s to form a single new solution F_{rs} . Each vertex in F_r is compared to the corresponding vertex in F_s . If one of the

vertices is determined to be dominant, that vertex and all associated edges are moved to F_{rs} . In the event neither vertex is dominant one of the vertices is chosen at random. The source and sink vertices are not used during crossover. The flow in and out of the graph is a by-product of balancing and maximizing the flow of the intermediate vertices.

The dominance of a vertex is determined by the energy level it possesses. The vertex with the lowest energy level is dominant. Energy is determined as follows:

$$\text{Energy} = k * |X_i| + |\min(I_{ic}, -O_{ic}) - \max(I_{if}, -O_{if})|$$

where k = balancing factor, a constant;

X_i = excess flow in the vertex;

$\min(I_{ic}, -O_{ic})$ = maximum flow capacity through the vertex;

$\max(I_{if}, -O_{if})$ = maximum flow through the actual vertex assuming it was balanced at the higher value;

A balanced vertex with a flow at maximum capacity will have an energy level of 0. The balancing factor k is used to emphasize balancing over maximizing the flow. For graphs without loops and bottlenecks a low k value increases the probability of finding the maximum flow. For complex graphs a higher setting will cause the algorithm to find a balanced solution at a reduced flow level.

The purpose of dominance in the crossover process is to increase the number of balanced low energy vertices in future generations. Initial experiments using standard crossover created new graphs which would have few if any balanced vertices. As a result these experiments failed to find a balanced graph. By using dominance and transferring complete vertices a graph which has several balanced nodes has a better chance of passing those balanced vertices to future generations. Additionally, vertices which were severely out of balance were removed from the population.

Step 6. Mutation

Mutation occurs as the edges of the dominant vertex are transferred to the new solution. The probability of mutation is variable depending upon the balance component of the energy level of the vertex. The probability of mutation is

determined as follows:

mutation rate = $x + \sqrt{\text{excess flow } i} / (\text{total flow } i)$;

where x = fixed mutation rate;

All edges attached to the vertex are subjected to the same probability of mutation. The fixed mutation rate x is small, usually around 0.02. The variable component of mutation fluctuates depending upon the balanced condition of the vertex. For a vertex with an energy level above zero mutation always occurs in the direction which will lower the energy level of the vertex. For a vertex with an energy level at zero mutation occurs in either direction at random. When mutation occurs edges are adjusted in increments of 1. Mutation is constrained by the upper and lower capacities of the edge. Mutation cannot take an edge below zero or above capacity.

Step 7 Assimilating the dominant vertex

Once a vertex is chosen it must be assimilated by the new graph. When an edge is added to the new graph a check is made to see if a value for that edge has previously been determined. If no previous value is available the value of the edge becomes:

edge = edge + mutation adjustment;

If a previous value was determined during the addition of a vertex earlier in the process the value of the edge is the average value of the two edges adjusted for mutation.

Edge = (old edge + new edge) / 2 + mutation adjustment;

The value of edge is an integer; when (old edge + new edge) / 2 has a fraction of 0.5, the result can be either truncated or rounded up to the closest integer. We chose this randomly, with a fixed truncation rate, such as 80 %, in which case the result is truncated 80 % and rounded up 20 % of the time.

This assimilation process allows the vertex to receive information about flow capacity from adjacent vertices which can be used by future generations to modify their internal flows. For example, the restricted flow caused by a bottleneck is propagated to surrounding vertices allowing these vertices to stabilize at the lower level after several generations.

4. EXPERIMENTAL RESULTS

The experiment was run for various graphs of different n as well as parameter values. The results of the experiment were affected by the parameter values as follows:

Balancing factor k (Step 5). Increasing k adds more weight to balancing the flow and places less emphasis on maximizing the flow. If k is too low, most of the flow would not be balanced; on the other hand, if k is too high, the flows will be balanced, but the optimality would be poor. When graphs are simple and easier for the program to balance a lower k value can be used. This allows the program to concentrate on finding the maximum flow. As the graphs increase in complexity balancing the graph becomes more difficult. The k value is increased to encourage balancing.

Mutation rate (Step 6). As the mutation rate increases the flow tends to increase while the number of generations to find a balanced solution also increases.

Truncation rate (Step 7). If the truncation rate is too low, the flows tend to go too high, exceeding the maximum flow resulting in graphs that cannot balance. A higher truncation rate reduces the flow level and the number of generations required to find a balanced solution.

Examples of our experiment are shown in Graphs 1 and 2, and the result in Table 1. Graph 1 contained 49 edges with no bottlenecks or loops. The maximum flow for Graph 1 is 90. Graph 2 contained 56 edges with loops but no bottlenecks. The maximum flow for Graph 2 is 91. Successful tests were performed on graphs containing bottlenecks but those results are not presented in this paper.

5. CONCLUSIONS

We have shown that the above genetic algorithm finds an optimal or near optimal solution for the maximum flow problem, with a reasonable number of iterations compared to other previous GA applications. The overall complexity of the current algorithm is not as good as when compared with the conventional approaches. A possible further study is to reduce the number of generations required to determine these solutions. Out of the two objectives, balancing and maximizing the flow, the difficult part is balancing. Convergence to maximum levels occurs rather rapidly. The remaining generations are spent trying to balance the solutions while slowly reducing the average flow in the population. Potential strategies for performance

improvements may be achieved by, for example, development of a better way of tuning the parameters, and modifications of the fitness function, reproduction, crossover and mutation schemes.

References

- R.K. Ahuja, J.B. Orlin and R.E. Tarjan, "Improved Time Bounds for the Maximum Flow Problem," *SIAM J. Comput.*, vol.18, no. 5, pp. 939-954, Oct., 1989.
- R.K. Ahuja and J.B. Orlin, "Distance-Directed Augmenting Path Algorithms for Maximum Flow and Parametric Maximum Flow Problems," *Naval Research Logistics*, vol. 38, pp. 413-430, 1991.
- N. Alon, "Generating Pseudo-Random Permutations and Maximum Flow Algorithms," *Information Processing Letters*, vol. 35, pp. 201-204, 1990.
- L. Davis, (Ed.). *Genetic Algorithms and Simulated Annealing*. Pitman, London, 1987.
- E.A. Dinic, "Algorithm for Solution of a Problem of Maximum Flow in a Network with Power Estimation", *Soviet Math. Dokl.*, vol. 11 (1970), pp. 1277-1280.
- L.R. Ford, Jr. and D.R. Fulkerson, *Flows in Networks*, Princeton, NJ: Princeton University Press, 1962.
- D.E. Goldberg, *Genetic Algorithms in Search, Optimization & Machine Learning*, Addison-Wesley, Reading, MA, 1989.
- J.R. Koza, *Genetic Programming*. MIT Press, Cambridge, MA, 1992.
- J. Marberg and E. Gafni, "An $O(n^3)$ Distributed Maximum Flow Algorithm", *Conf. on Inf. Sci. & Sys.*, Princeton, NJ, pp. 478-482, 1984.
- G. Mazzoni, S. Pallottino and M.G. Scutella, "The Maximum Flow Problem: A Max-Preflow Approach," *European Journal of Operations Research*, vol. 53, pp. 257-278, 1991.
- J.A. McHugh, *Algorithmic Graph Theory*, Englewood Cliffs, NJ: Prentice-Hall, 1990, Chapter 6.
- Z. Michalewicz, A step toward topology of communication networks, *Proceedings of the SPIE - The International Society for Optical Engineering*, vol. 1470, pp. 112-122, 1991.
- A. Segall, "Decentralized Maximum-Flow Protocols", *Network*, vol. 12, No. 3, pp. 213-230, 1982.
- Y. Shiloach and U. Vishkin, "An $O(n^2 \log n)$ Parallel MAX-FLOW Algorithm", *Journal of Algorithm*, vol. 3, 1982, pp. 128-146.

R.E. Tarjan, *Data Structures and Network Algorithms*, Philadelphia, PA: Society for Industrial and Applied Mathematics, 1983.

D.Y. Yeh and T. Munakata, "A Maximal Flow Algorithm in a Distributed Layer Network," *Proceeding of International Computer Symposium, Tainan, Taiwan*, Dec., 1986, 1221-12227.

Balancing factor $k = 1.4$

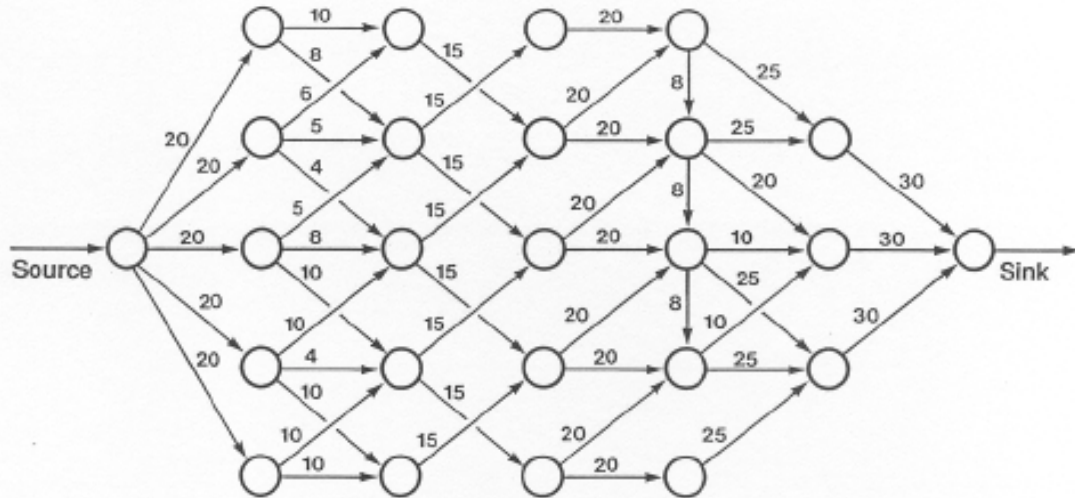
Run	Graph 1		Graph 2	
	Found	Maximum Generation	Found	Maximum Generation
1	90	54	89	126
2	90	69	87	85
3	90	64	-	200
4	90	68	-	200
5	90	86	88	137

Balancing factor $k = 2.0$

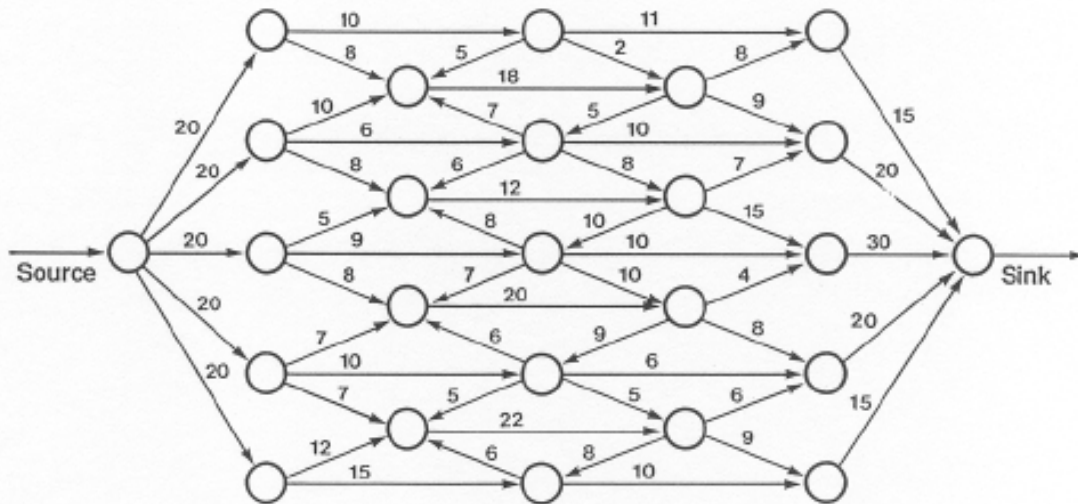
Run	Graph 1		Graph 2	
	Found	Maximum Generation	Found	Maximum Generation
1	90	77	83	111
2	89	49	83	71
3	89	50	86	135
4	90	53	86	135
5	90	36	85	145

Table 1: Experimental results

Population size = 50, Fixed mutation rate, $x = 0.03$, Truncation rate 80 %. A Maximum Found of "-" indicates that the program was terminated at a predetermined generation limit = 200 without finding a balanced flow.



Graph 1. 25 vertices. Maximum flow = 90.



Graph 2. 25 vertices. Maximum flow = 91.

Figure 1. Maximum flow problems, where the numbers at the edges are the flow capacities.