Performance Enhancement of Routing Algorithm Based on PSO

Allam MAALLA

Research Institute of Open Data Commercial Application, School of Information Technology and Engineering, Guangzhou College of Commerce, Guangzhou, China

Keywords: Shortest path routing, Genetic algorithms, Particle swarm optimization, Q-PSO.

Abstract. Telecommunication networks have become a special focus of research in both academia and industry due to the unprecedented growth of the Internet popularization. This paper focuses on the enhancement of routing algorithm by using an intelligent algorithm approach to the Q-routing problem. The Q-routing algorithm developed by adding intelligent represents determined state for each link to serve as constraint and contributes in choosing the appropriate path. The computer networks considered are packet switching networks, modeled as directed graphs where nodes represent servers, hosts or switches, while bi-directional and symmetric arcs represent full duplex communication links. By adding PSO into its proposed algorithm, which is based on shortest delivery time, focuses on improving quality of service by depending on defined state of shortest delivery time to filter out any inappropriate path.

Introduction

The shortest path (SP) problem concerns about finding the shortest path from a specific origin to a specified destination in a given network while minimizing the total cost associated with the path. This problem has been investigated extensively. The well-known algorithms for solving it include the Bellman’s dynamic programming algorithm for directed networks, the Dijkstra labeling algorithm and Bellman–Ford successive approximation algorithm for networks with non-negative cost coefficients. These traditional algorithms have major shortcomings; firstly, they are not suitable for networks with negative weights of the edges, i.e., in communication networks, the link weights represent the transmission line capacity and negative weights correspond to links with gain rather than loss. Secondly, the algorithms search only for the shortest route, but they cannot determine any other similar/non-similar short routes (which is commonly referred to as the kth SP problem). Thirdly, they exhibit high computational complexity for real-time communications involving rapidly changing network topologies such as wireless ad hoc networks.

Related Work

New techniques have been continuously explored; Artificial neural networks (ANN) have been examined to solve the SP problem relying on their parallel architecture to provide a fast solution. However, the ANN approach has several limitations. These include the complexity of the hardware, which increases considerably with increasing number of network nodes; at the same time, the reliability of the solution decreases. Secondly, they are less adaptable to topological changes in the network graph including the cost of the arcs. Thirdly, the ANNs do not consider sub-optimal paths. Among other approaches for this problem, the powerful evolutionary programming techniques have considerable potential to be investigated in the pursuit for more efficient algorithms because the SP problem is basically an optimal search problem. In this direction, genetic algorithm (GA) has shown promising results. The most recent notable results have been reported in. Their algorithm shows better performance compared to those of ANN approach and overcomes the limitations mentioned above.
Shortest Path Routing

The idea is to build a graph of the subnet, with each node of the graph representing a router and each arc of the graph representing a communication line (called a link). To choose a route between a given pair of routers, the algorithm just finds the shortest path between them on the graph.

One-way of measuring path length is the number of hops. Using this metric, the paths ABC and ABE in Fig.1 are equally long. Another metric is the geographic distance in kilometers, in which case ABC is clearly much longer than ABE. Having examined each of the nodes adjacent to A, it will examine all the tentatively labeled nodes in the whole graph and make the one with the smallest label permanent, as shown in Fig.1 (b). This one becomes the new working node.

After all the nodes adjacent to the working node have been inspected and the tentative labels changed if possible, the entire graph is searched for the tentatively labeled node with the smallest value. This node is made permanent and becomes the working node for the next round. Fig.1 shows the first five steps of the algorithm.

To see why the algorithm works, look at Fig.1(c). At that point it may have just made E permanent. Suppose that there were a shorter path than ABE, say AXYZE. There are two possibilities: either node Z has already been made permanent, or it has not been. If it has, then E has already been probed (on the round following the one when Z was made permanent), so the AXYZE path has not escaped our attention and thus cannot be a shorter path. Now consider the case where Z is still tentatively labeled. Either the label at Z is greater than or equal to that at E, in which case AXYZE cannot be a shorter path than ABE, or it is less than that of E, in which case Z and not E will become permanent first, allowing E to be probed from Z.

Genetic Algorithms

Genetic Algorithms were invented to mimic some of the processes observed in natural evolution. Many people, biologists included, are astonished that life at the level of complexity that we observe could have evolved in the relatively short time suggested by the fossil record. The idea with GA is to use this power of evolution to solve optimization problems.

Genetic Algorithms (GAs) are adaptive heuristic search algorithm based on the evolutionary ideas of natural selection and genetics. As such they represent an intelligent exploitation of a random search used to solve optimization problems. Although randomized, GAs are by no means random, instead they exploit historical information to direct the search into the region of better performance within the search space. It is better than conventional AI in that it is more robust. Unlike older AI systems, they do not break easily even if the inputs changed slightly, or in the presence of reasonable noise.

GAs simulates the survival of the fittest among individuals over consecutive generation for solving a problem. Each generation consists of a population of character strings that are analogous to the chromosome that we see in our DNA. Each individual represents a point in a search space and a possible solution. The individuals in the population are then made to go through a process of
evolution. GAs are based on an analogy with the genetic structure and behavior of chromosomes within a population of individuals using the following foundations:

**Search Space**

A population of individuals is maintained within search space for a GA, each representing a possible solution to a given problem. Each individual is coded as a finite length vector of components, or variables, in terms of some alphabet, usually the binary alphabet \{0,1\}. To continue the genetic analogy these individuals are likened to chromosomes and the variables are analogous to genes. Thus a chromosome (solution) is composed of several genes (variables). A fitness score is assigned to each solution representing the abilities of an individual to `compete'. The individual with the optimal (or generally near optimal) fitness score is sought. The GA aims to use selective `breeding' of the solutions to produce `offspring' better than the parents by combining information from the chromosomes as shown in (Fig. 2).

![Figure 2. Search space.](image)

The GA maintains a population of n chromosomes (solutions) with associated fitness values. Parents are selected to mate, on the basis of their fitness, producing offspring via a reproductive plan. Consequently highly fit solutions are given more opportunities to reproduce, so that offspring inherit characteristics from each parent. As parents mate and produce offspring, room must be made for the new arrivals since the population is kept at a static size. New generations of solutions are produced containing, on average, more good genes than a typical solution in a previous generation. Each successive generation will contain more good `partial solutions' than previous generations. Eventually, once the population has converged and is not producing offspring noticeably different from those in previous generations, the algorithm itself is said to have converged to a set of solutions to the problem at hand.

**Particle Swarm Optimization**

The PSO belongs to the class of direct search methods used to find an optimal solution to an objective function (aka fitness function) in a search space. Direct search methods are usually derivative-free, meaning that they depend only on the evaluation of the objective function. The particle swarm optimization algorithm is simple, in the sense that even the basic form of the algorithm yields results, it can be implemented by a programmer in short duration, and it can be used by anyone with an understanding of objective functions and the problem at hand without needing an extensive background in mathematical optimization theory.

The PSO is a stochastic, population-based computer algorithm modeled on swarm intelligence. Swarm intelligence is based on social-psychological principles and provides insights into social behavior, as well as contributing to engineering applications.

Particle swarm optimization is inspired by this kind of social optimization. A problem is given, The swarm is typically modeled by particles in multidimensional space that have a position and a velocity. These particles fly through hyperspace (i.e., R^n) and have two essential reasoning capabilities: their memory of their own best position and knowledge of the global or their neighborhood's best. In a minimization optimization problem, problems are formulated so that "best" simply means the position with the smallest objective value. Members of a swarm communicate good positions to each other and adjust their own position and velocity based on these good positions. So a particle has the following information to make a suitable change in its position and velocity:
• A global best that is known to all and immediately updated when a new best position is found by any particle in the swarm.
• Neighborhood best that the particle obtains by communicating with a subset of the swarm.
• The local best, which is the best solution that the particle has seen.

The basic idea of the classical particle swarm optimization (PSO) algorithm is the clever exchange of information about the global and local best values mentioned above. Let us assume that the optimization goal is to maximize an objective function \( f(r) \). Each particle will examine its performance through the following two views. Each potential solution is also assigned a randomized velocity, and the potential solutions, called particles, correspond to individuals. Each particle in PSO flies in the D-dimensional problem space with a velocity dynamically adjusted according to the flying experiences of its individuals and their colleagues. The location of the \( i^{th} \) particle is represented as \( X_i = [x_{i1}, x_{i2}, \ldots, x_{iD}] \), Where \( x_{i1}, x_{i2}, \ldots, x_{iD} \) are the lower and upper bounds for the \( d^{th} \) dimension, respectively. The best previous position (which gives the best fitness value) of the \( i^{th} \) particle is recorded and represented as \( P_i = [p_{i1}, p_{i2}, \ldots, p_{iD}] \), which is also called \( x_{gbest} \). The index of the best particle among all the particles in the population is represented by the symbol \( (P_{gbest}) \). The location \( (P_{gbest}) \) is also denoted by \( x_{gbest} \). The velocity of the \( i^{th} \) particle is represented by \( v_{id} \) and is clamped to a maximum velocity which is specified by the user. The particle swarm optimization concept consists of, at each time step, regulating the velocity and location of each particle toward its \( x_{gbest} \) and \( x_{lbest} \) locations according to (1) and (2), respectively

1. \[ v_{id}^{t+1} = v_{id}^{t} + c_1 r_1 (p_{id} - x_{id}) + c_2 r_2 (x_{gbest} - x_{id}) \]
2. \[ x_{id}^{t+1} = x_{id}^{t} + v_{id}^{t+1} \]

where \( w \) is the inertia weight; \( c_1, c_2 \) are two positive constants, called cognitive and social parameter respectively \( d = 1, 3, \ldots, D; i = 1, 3, \ldots, m \) and \( r \) is the size of the swarm; \( r_1, r_2 \) are two random numbers, uniformly distributed in \([0,1]\); and \( n = 1, 3, \ldots, N \) denotes the iteration number, \( N \) is the maximum allowable iteration number.

**Proposed Algorithm**

Q-Learning is a model free algorithm that learns from the delayed reinforcements and it is one of the easiest approaches to implement in reinforcement learning, thus one of the most popular. Q-learning is applied to the routing problem in Q-Routing algorithm, where routing table in the distance vector algorithm is replaced by the table of estimations Q values based on the link delay.

In the Q-Routing same routing policies are used as in the distance vector routing algorithm. Q-function is used to update routing table entries in Q-Routing. However, it has been suggested that neural networks can be used for approximating the Q-function by incorporating diverse parameters of the network such as time delays, queue lengths and etc. In Q-Routing, when x sends a packet to the node d via its neighbor y it receives y’s estimated remaining trip times to the destination d. Then it selects the neighbor with the smallest time Eq.3. Q-Routing uses forward exploration to update Q values in the routing table that represents the delivery time estimation to the given destination Eq.4:

\[ Q(y,d) = \min_{z \in N(y)} Q_y(z,d) \]

\[ \Delta Q_x(y,d) = n_t [Q_x(D,n) + q_x(y,d) - Q_x(y,d)] \]

Q(y,d) is the new estimation value for node x to destination d via the neighbor node y. This new estimation is calculated by subtracting old estimation value \( Q_x(y,d) \) from the sum of the estimation time for packet travelling from node y to destination d via neighbor z \((Q_y(z,d))\) and current queue
delay for the packet in node x (qx). $\eta^x$ is the learning rate parameter. In Figure 3, x updates its $Q_x(y,d)$ value pertaining to the remaining path of packet $p$ via node y.

![Figure 3. Example of routing path.](image)

**Routing Path Based on PSO**

A Particle of the proposed PSO consists of sequences of positive integers that represent the IDs of nodes through which a routing path passes. Each locus of the particles represents an order of a node in a routing path. The particle of first locus is always reserved for the source node. The length of the particles is variable, but it should not exceed the maximum length, which is equal to the total number of nodes in the network, since it never needs more than number of nodes to form a routing path. A particle (routing path) encodes the problem by listing up node IDs from its source node to its destination node based on topological information database (routing table) of the network. An example of particles (routing path) encoding from node to node is shown in Figure 1. The particles are essentially a list of nodes along the constructed path, (P). The first element of the each particle encodes the source node, and the remaining element of particles is randomly or heuristically selected from the nodes connected with the source node. Then $P_{best}$ will present the shortest path and $g_{best}$ will list the nodes in the path. This process continues until the destination node is reached.

The performance parameters for the simulations are:

1. Average packet delay: is the average delay a packet experiences will depend on shortest route from source to destination.
2. Average throughput per packet: is the average number of packets being forwarded by a node for the duration of the simulation.

**Implementation**

PSO based on Q-Routing algorithm were implemented in the following environment:

Assigning every node to a different process both on the same and distributed machines simulates parallel behavior as shown in Fig.3.

Random uniform traffic distribution is used with varying packet creation rates.

For each simulation, packet generation is stopped after creating 2000 packets per node and simulation is stopped after all packets are arrived to their destinations or detected and deleted from the network.

Every simulation is run 3 times and the average of the results is used for accuracy.

A random link fails every 6 seconds for a period of 5 seconds and traffic is directed to other neighbors. Packet payload is fixed (512 Kbytes).

From Fig.4, it can be seen that as the system load increases average packet delay increases for all algorithms. Therefore, for Q-Routing algorithm, the delivery times are lower compared to the Q-PSO algorithm as it was expected.
For the Q-PSO algorithm, the delivery time at high loads was reasonably high compared to Q-Routing, which can be accounted for the packet overhead, which makes the packet to carry the learning update along with the data.

Finally, these figures are only valid for our test environment and effect of learning rate needs to be investigated in the future for different systems for a more general figure rather than using constant values based on experience.

**Conclusion**

The proposed method is based on a network representation enabling to match each network configuration, with a heuristic approach in order to help to find the path between any node pair. The Software presented here is based in original on Q-routing algorithm that means if there is no heuristic state entered through entrance of distance value, it will find path according to Q-routing algorithm. The use of another parameter to find the path makes this software able to be applied to achieve quality of service, just like filter out inappropriate path that doesn’t satisfy QoS requested, then use Q-PSO routing to find best path through remaining path. If one link fails, then this algorithm will find another path to pass packet through. The continuously assess the optimal path within the network so any packet needs to be forwarded, the route is always known and the path can currently be used.

**References**


