A Distributed Search (Best-Path) Algorithm based on Local Communication

João Neto[1], and **Helder Coelho**[1]

[1] Departamento de Informática, Faculdade de Ciências da Universidade de Lisboa, C5 – Piso 1, 1700 Lisboa, PORTUGAL,
{jpn, hcoelho}@di.fc.ul.pt

Abstract.
Finding the best path between two nodes on an arbitrary communication network is a complex task. Instead of using a central processing unit to keep network topology and compute the best path by request, herein all nodes will react when a network node/arc is inserted/deleted, and eventually producing other reactions on their immediate neighbors, causing a temporary chain reaction until the net reaches a new equilibrium. This individual behavior will produce a global organization of the net (seen as a society of interconnected nodes), where each node will keep the best paths to reach the others, without the need of a central organizational node, thus reducing the expected algorithm complexity, diluting it through all independent agents within the system.

1 Introduction

The best path problem is a common optimization problem, appearing in many different scientific areas, like in traffic and transportation simulations. There is no best algorithm to solve all specific best path problems (check [1] and [7] for surveys and evaluations of several algorithms). The problem in hands may need a very detailed graph with several constraints, like congestion after a certain flow is reached, within a dynamic network topology, including flow priorities, and so on… There are several attempts on the Distributed Artificial Intelligence scientific area, which includes: (a) Emergent models of reactive agents, cf. [10], [11] and [14]. One of these model's inspirations lies on biology, namely how social insects deal with path solving problems (like the use of pheromones in ant paths). (b) Economic concepts, like the use of an utility function and the subsequent notion of market, dealing with the available resources, allocation capacity to those resources, trade and prices, cf. [3], [8], [9] and [13] for several works on this field. (c) Active nets, where each message is no longer seen as a passive set of data, but has an active process capable of decision making. Each node flips into an active holder of valuable information to be accessed by those dynamic messages, check [6] and [12] for some surveys.

We will present a different view from these points. An agent, represented herein by a node on the communication network, is not a collector of information (like in *(a)*), or a buyer of a certain established path (like in *(b)*), but it is able to make a decision based solely on what it knows about the region it lies inside the network. It will be also able to change its internal status accordingly to the outside messages it receives from other nodes (e.g., some distant node was disconnected) or from the environment itself (e.g., some direct connection was removed).

2 Central vs. Distributed Approaches

The Dijkstra standard algorithm to find the best path of a graph (cf. [2]) has a low polynomial complexity given by $O(n^2)+O(m)$, where n is the number of nodes, and m the number of connections. Floyd suggests another known algorithm with polynomial complexity $O(n^3)$. The matrix P says that for an initial node i to go to final node j, the path continues to its immediate neighbor k. Let's assume a simple graph with n nodes defined by the adjacency matrix G. The algorithm follows:

```
for (k = 1; k <= n; k++)
  for (i = 1; i <= n; i++)
    for (j = 1; j <= n; j++)
      if (G[i][j] > (G[i][k] + G[k][j])) {
        G[i][j] = G[i][k] + G[k][j];
        P[i][j] = k;
      }
```

There are two relevant questions about these standard algorithms. The first is that the criteria that define what is distance may change the problem intrinsic complexity. On ATMs, the concern about transmission rate, communication delay, lost package percentage, transforms the well-behaved polynomial problem into an NP-complete one. And this is the general case for actual problems, and not an exception…

The second question focus on the fact that this type of solutions needs a central processing unit or server. This unit knows all network topology – which includes all connections and related information about them. For certain problems, where the network is too large or too dynamic, this may be even impossible to have.

On a centralized environment, the server receives messages informing it of any network change, like the insertion or removal of a node or connection. After running the algorithm to find the new short paths, it must communicate to all nodes informing them of the eventual new routes. So, if the network is too dynamic, there is an intense message flow between the server and its nodes, creating a new delay caused by those potential message queues.

On a distributed environment, the information is shared in regional areas consisting of sets of nodes. Each node is only concerned about what happens to itself (e.g., one of its direct connections is removed) and how its immediate neighbors behave during time. So, each node is an independent agent, dealing with the problems of the outside ambient – which is the sum of all possible events plus all those adjacent nodes. If fact, there is no restriction that says that every node is equal to the next. One assumption made is that the information received by another node is true, according to what the sender knows on that particular moment. Also, each node is parsimonious on the number of sent messages. It will avoid any type of broadcast and only focus on those nodes that it thinks will profit from the actual information.

3 Network Topology

The net topology is defined by the basic bi-directional graph, with labeled arcs. These labels consist of non-negative numbers representing the specific cost of a message sending between the two nodes defining that same arc. This cost gives the metric to define what a best path means (usually representing time delay or some associated economic cost).

A graph sample could be:

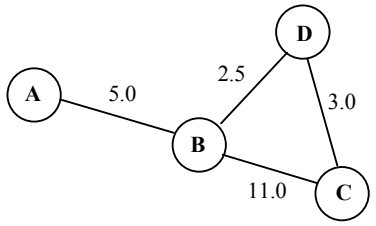

Figure 1. A communication network.

This network is composed of 4 nodes and 4 connections. The cost to travel from A to B (or B to A) is 5.0 units. There are two possible paths to go from A to D, namely, A→B→D with cost 7.5, and A→B→C→D with cost 19.0.

A relevant difference between the centralized and the distributed approach is how to represent this connection matrix defining the graph. For the former approach, a bi-dimensional vector is used (like in the Floyd algorithm), while in the later, each node will keep a one-dimensional vector (called the pathway vector), which keeps the specific row of the matrix concerning itself.

4 Algorithm Description

Our approach will see each node as an agent capable of dealing with several distinct problems, namely, how to sent information to a certain destiny and how to solve all possible and unexpected changes of network topology. To handle it, each node/agent will be equipped with a specific data structure (the pathway vector) and algorithm, which are described on this section. Its internal structure can be described with the following diagram:

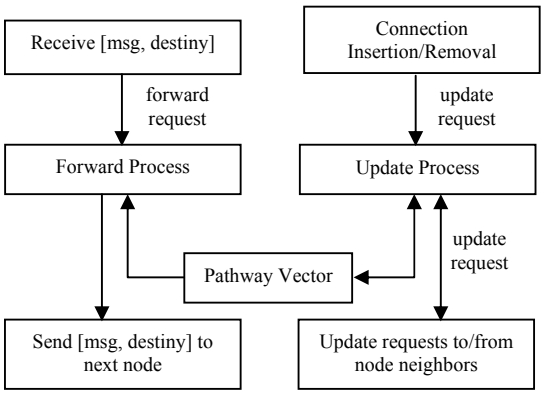

Figure 2. The agent internal structure.

All agents through the network are alike, but, as said, this is not a requirement. However, the algorithm is based on the following assumptions:

(a) The paths are not known *a priori*. They are computed and updated when a change occurs on the network. This means, that a decision to send information to a certain destiny is a consequence of the network actual state. The decision process adapts to the network state.

(b) The complete path between A and B, is neither known to A nor B (except if it is a transfer between adjacent nodes). The sender only knows the next step of the way, trusting that the node on the other side knows where to send the required information.

(c) The system is proactive, not reactive. Problems must be solved before they occur! This implies that – at a certain primitive level – all nodes are cognitive, not reactive ones.

(d) Nodes should know how its regional network area is defined (this means, all adjacent nodes and they associated connection costs).

Accordingly to [16], an agent *coordination* is defined by a set of agents working together on a specific task (or limited set of tasks), while an *organization* is a more stable set of communication protocols between a society of agents, to handle a set of tasks and not necessarily related.

In the present problem, a coordination is an episodic situation where an information package travels between a set of nodes, from the sender until it reaches the receiver. If the pathway vector is well updated, a coordination episode is trivial. Check if the message if for itself. If it is, process it. Otherwise, look into the vector, what is the next step to send the package towards destiny.

Herein the protocol, defining the organization that unites all nodes, is represented by the set of valid messages that can flow between any pair of nodes. The set of valid messages are: (i) Deactivation (auto-removal); (ii) Connection insertion to a new neighbor; (iii) Connection removal from an immediate neighbor; (iv) Cost update of a direct connection; (v) Question about the path and cost to a certain node.

To keep and process these messages, each node has a pathway vector, where it saves the actual best path and cost for each available node. When a message is received, the node must update (if necessary) the pathway vector to maintain it consistent with the received information. Also, if it finds that the information may be useful to some neighbor, it will forward to it. It is through this filtering process that the network is able to reach equilibrium and avoid potential deadlocks (e.g., cycles where a set of nodes sends 'useful' information to each other in infinite loops). There are 6 different events, some of those responsible for these organizational-driven messages.

(a) Node activation – this event implies that a new node was added to the network. Since the new node comes with no connection, there are no messages produced.

(b) New direct connection between node A and node B – both nodes will sent to the remaining neighbors this node information together with the associated cost. They will also update their own pathway vectors.

(c) New better indirect connection – if an agent receives a new pair (connection, cost) that is better than the one saved on his pathway vector, the node updates it, and sends to all remaining neighbors this new update.

(d) Node deactivation – the exiting node informs all neighbors of this fact. Each node receiving this information, and realizing that it needed that node, will ask the other neighbors for a second option (and getting answers if any neighbor has some).

(e) Direct connection removal – both nodes will send messages to all other neighbors informing about the event. After that, they will ask those same neighbors for a second option.

(f) Indirect connection removal – the node, receiving this information, will check if that connection was used by him (i.e., if it used the immediate neighbor that sent it the message). If so, it will ask the other neighbors for second options; otherwise, it does nothing.

Since each node only sends messages if something improved its pathway vector, the message rate, after an initial sudden increase of message exchanges, will slow down and extinct, after all nodes share that information.

As an example, let's assume a network like the one in fig. 1, and insert a new node between nodes A and D, thus obtaining the following network:

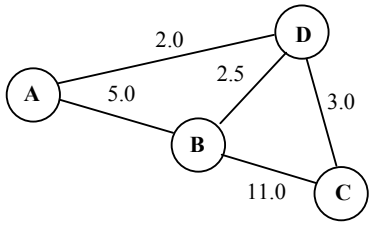

Figure 3. Inserting a new connection.

The set of messages and updates produced follows:

```
[A]     Update connection to D with delay 2 by D
[D]     Update connection to A with delay 2 by A
[A→B]   New connection to D with delay 2
[A→D]   Connection to B with delay 5
[A→D]   Connection to C with delay 10.5
```

```
[D→B]  New connection to A with delay 2
[D→C]  New connection to A with delay 2
[D→A]  Connection to B with delay 2.5
[D→A]  Connection to C with delay 3
[B]    Update connection to A with delay 4.5 by D
[B→C]  New connection to A with delay 4.5
[C]    Update connection to A with delay 5 by D
[C→B]  New connection to A with delay 5
[A]    Update connection to B with delay 4.5 by D
[A]    Update connection to C with delay 5 by D
[A→B]  New connection to C with delay 5
```

At this moment, the nodes that received the last messages do not have the necessity to update their pathway vectors (the sent information is already known), and so the message rate drops to zero. The network has reached a new equilibrium. If now, we remove that same connection (between A and D), the messages and updates would be:

```
[A→B]  Indirect connection removal with D
[D→C]  Indirect connection removal with A
[A→B]  Indirect connection removal with C
[D→B]  Indirect connection removal with A
[B]    Connection update to A with direct link with delay 5
[B→A]  New connection to C with delay 5.5
[B→D]  New connection to A with delay 5
[A]    Creating new connection to C by B with delay 10.5
[D→C]  Have you a connection with A? [no, but searching...]
[D→B]  Have you a connection with A? [yes, using A]
[D]    Creating new connection to A by B with delay 7.5
[B→A]  New connection to D with delay 2.5
[B→C]  Indirect connection removal with A
[A→B]  Have you a connection with C? [yes, using D]
[B→D]  New connection to A with delay 5
[A]    Creating new connection to D by B with delay 7.5
[B→C]  New connection to A with delay 5
[A→B]  Have you a connection with D? [yes, using D]
[C]    Creating new connection to A by B with delay 10.5
[B→A]  New connection to C with delay 5.5
[D→C]  New connection to A with delay 7.5
[C→D]  New connection to A with delay 10.5
[A]    Updating conection to C by B with delay 10.5
[C→B]  Indirect connection removal with A
[C→B]  Have you a connection with A? [yes, using A]
[D→C]  New connection to A with delay 7.5
[B→C]  New connection to A with delay 5
[B→A]  New connection to D with delay 2.5
```

After this last message, the network reached the previous equilibrium before the A↔D connection insertion.

The faster the connection, the more messages the network will eventually exchange between interested nodes. If a slow connection is inserted, the number of messages will be lower, specifically between the pair of nodes that received the direct connection and their immediate neighbors (which will not send the information any further, since the node is not ‘competitive’).

5 Simulation Results

To simulate this network (society) of nodes (agents) it was used the JAVA software SACI (cf. [4]), which is an agent-oriented workbench, aimed to ease the programmer task on simulation problems like this one. The main focus of SACI is to enable distributed agents to communicate in easier ways. For more technical information (including package download) and recent updates, check *http://www.lti.pcs.usp.br/saci*.

SACI features include communication among KQML messages. Agents can send messages to others agents just using receiver's name – the location in the network is made transparent to the sender by a white pages service. Agents might know others by a yellow pages service (they can register their services and query to find out what services are offered by others). And since SACI is made with the JAVA programming language, agents can be implemented as applets and run on web browsers.

The next table shows the average results of ten experiences for each chosen network topology. The topology is defined by a number of nodes and an average connection per node. For e.g., the second line of the table, show results for networks with a total of four nodes and an average of two connections per node (i.e., the network had a total of eight connections).

Both the centralized Floyd algorithm and our distributed algorithm are presented in competition. The first column for each algorithm represents the computation cycles needed for each independent processing unit. The second column for the centralized model shows the number of sent messages. For the distributed model, the number pair represents the number of sent messages by the entire system, and the average messages per node.

5.1 Complexity Issues

Probably the most relevant feature on Table 1 is the growth of computational steps needed for problem solving. For the Floyd approach, the proposed algorithm, as already said, has complexity $O(n^3)$, which is not bad (algorithms with polynomial complexity are among the worst cases of 'fast' algorithms). However, on a network with 10000 nodes, the computations would rise to a million of millions processing cycles.

On the distributed case, the algorithm complexity is divided into two different aspects: communication and internal computation. The former, in the worst case (all nodes are directly connected to each other) the communication complexity is quadratic. However, this scenario is more an exception than a rule for large networks (where the average number of arcs per node, is much smaller than the total number of nodes on the network). As the simulation results on the next section point us, the communication complexity seems to be O(n.m), with *n* nodes and an average of *m* arcs per node.

Internal Computation has complexity O(m), because of the need to access the pathway vector. Also, in the worst case, the complexity is O(n). Adding both complexities, we have O(n.m)+O(m). In the worst case (where $m=n$), we get $O(n^3)$.

The complexity of the centralized model is diluted between all nodes inside the network. The global equilibrium is reached based solely on a finite set of local exchanges consisting of regional information.

Table 1. Simulation Results

Topology	Centralized		Distributed	
	1	2	1	2
3 – 2	27	4,8	3	4,3 – 1,4
4 – 2	64	8,0	4	7,2 – 1,8
5 – 2	125	11,6	5	15.8 – 3,2
8 – 3	512	15,1	8	65,4 – 8,2
10 – 3	1000	21,0	10	143,8 – 14,4
11 - 4	1331	19,0	11	155,2 – 14,1
15 – 4	3375	29,3	15	389,4 – 26,0

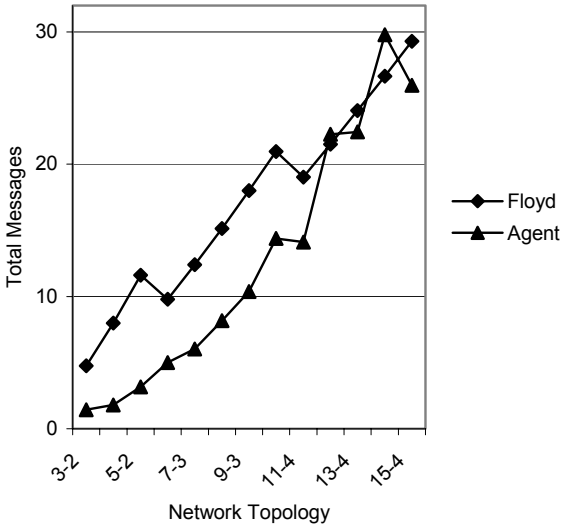

Figure 4. Graphical representation of Table 1.

6 Conclusion

In our distributed model there is no concern of global consistency (which would be impossible to keep, for sufficient large problems…) but only of local consistency, in opposition to the central approach. Like for the speed of light on our Universe, the information will flow, as fast as possible, to cover all nodes of the system, but meanwhile, a node far away may have an erroneous description of a distant area. However, since information cannot flow faster than one message per interconnection per time unit, it will find a consistent area sooner or later, when traveling towards its destination, which will send it to the proper place. This implies that in certain moments on certain areas, the best path is not known, but this is a conjectural situation not a structural one. The system dynamics assures that a new equilibrium is found, and the best path is known in the nearest possible future.

Several directions are available for future work. In giving more cognitive tools to each agent, it is possible to achieve 'smarter' networks where, for instance, each node can propose changes in topology (if possible) if it learns that a certain new connection would optimize its performance, or an obsolete connection is no longer used. Another investigation line is to have in each node, a more regional representation of its near but not immediate neighbors, so that the node can have more independent action if some connection is inserted/removed, in order to decrease the number of sent messages due to synchronization purposes.

Acknowledgements

We would like to thank the technical and financial support given by LabMAC and the Portuguese R&D agency Fundação de Ciência e Tecnologia.

Also, we would like to thank the financial support of the Brazilian R&D agency CAPES and the Portuguese R&D agency ICCTI, within the joint Portuguese-Brazilian project SCCI (Computational and Intelligent Control Systems).

References

1. Cherkassky, B., Goldberg, A. and Radzik. T., *Shortest Paths Algorithms: Theory and Experimental Evaluation*, Mathematical Programmming, 73, 1996, 129-174.
2. Dijkstra, E., *A Note on Two Problems in Connection with Graphs*, Numerical Mathematics, [1], 1959, 269-271.
3. Gibney, M. and Jennings, N., *Market Based Multi-Agent Systems for ATM Network Management*, Proc. 4th Communications Networks Symposium, Manchester, UK, 1997.
4. Hübner, J. and Sichman, J., *SACI: A Tool for Implementation and Communication between Agents* (in Portuguese), International Joint Conference IBERAMIA/SBIA 2000, Atibaia (São Paulo, Brazil), 2000.
5. Jennings, B., Brennan, R., Gustavsson, R., Feldt, R., Pitt, J., Prouskas, K., and Quantz, J., *FIPA-compliant agents for real-time control of Intelligent Network traffic*, Computer Networks, [31] 19, 1999, 2017-2036.
6. Kramer, K., Minar, N., and Maes, P., *Tutorial: Mobile Software Agents for Dynamic Routing*, www.media.mit.edu/nelson/research/routes/, 1999.
7. Pallottino, S. and Scutellà, M.; *Shortest path algorithms in transportation models: classical and innovative aspects*, in (P. Marcotte and S. Nguyen, eds.) Equilibrium and Advanced Transportation Modelling, Kluwer, 1998, 245-281.
8. Patel, A., Prouskas, K., Barria, J., and Pitt, J., *Load Control Using a Competitive Market-Based Multi-agent System*, Lecture Notes in Computer Science 1774, 2000, 239-254 ,
9. Prouskas, K., Patel, A., Pitt, J., and Barria, J., *A Multi-agent System for Intelligent Network Load Control Using a Market-based Approach*, The Third International Conference on Multi-Agent Systems (ICMAS '98), 1998, 231-238.
10. Schoonderwoerd, R., Holland, O., Bruten, J., and Rothkrantz, L., *Ant-based Load Balancing in Telecommunications Networks*, Adaptive Behavior, [5] 2, 1997, 169–207.
11. Subramanian, D., Druschel, P., and Chen, J., *Ants and Reinforcement Learning: A Case Study in Routing in Dynamic Networks*, Proc. of IJCAI'97, 1997, 832-838.
12. Tennenhouse, D., Smith, J., Sincoskie, W., Wetherall, D., and Minden, G., *A Survey of Active Network Research*, IEEE Communications Magazine, [35] 1, 1997, 80-86.
13. Wellman, M., A *Market-Oriented Programming Environment and its Applications to Distributed Multicommodity Flow Problems*, Journal of AI Research, [1] 1, 1993, 1-22.
14. White, T. and Pagurek, B., *Towards Multi-Swarm Problem Solving in Networks*, The Third International Conference on Multi-Agent Systems (ICMAS '98), 1998, 333-340.
15. Willmott, S. and Faltings, B., *Active Organisations for Routing*, Lecture Notes in Computer Science Series 1653, 1999, 262-273.
16. Willmott, S. and Faltings, B., *The Benefits of Environment Adaptive Organisations for Agent Coordination and Network Routing Problems*, The Fourth International Conference on Multi-Agent Systems (ICMAS-2000), 2000, 333-340.