

Calculating optimal decision using Meta-level agents for Multi-Agents in Networks

Anne Håkansson¹ and Ronald Hartung²

¹Department of Information Science, Computer Science,
Uppsala University,
Box 513, SE-751 20,
Uppsala, Sweden
Email: Anne.Hakansson@dis.uu.se

²Department of Computer Science,
Franklin University,
201 S. Grant Avenue,
Columbus, Ohio 43215, USA
Email: Hartung@franklin.edu

Abstract. In spatial graphs with a vast number of nodes, it is difficult to compute a solution to graph optimisation problems. We propose an approach using meta-level agents for multi-agents in a network to calculate an optimal decision. The network contains nodes and arcs wherein the agents are information carriers between the nodes and, since there is one agent per arc, the agents are statically located. These agents, operating at a ground level, communicate with a comprehensive agent, operating at a meta-level. The agents at the meta-level hold information computed by the ground-level agents, but also include ground-level agents' special conditions. As an example, we apply the work to the travelling salesman problem and use a map, with cities and roads, constituting the network where the information about the roads is carried in the meta-level agents. For multi-agents in maps, we use parallel computing. The parallel computing is at the ground-level agents' level and simulates geographical information systems to provide the agents with environmental information.

1 Introduction

Spatial networks can, in general, be used to describe any network in which links or arcs connect nodes. In physical spaces, spatial networks can be derived from maps, using features like the road segments. Spatial networks can also represent complex social networks (Internet) and computer networks. To work with these different networks, there have been proposals of using a range of intelligent agents [13].

A challenge with network problems is finding and extracting information in the network within an acceptable time bound. As solutions, there have been attempts at applying effective search strategies, e.g., heuristics, A*, Best-first search, breadth-first, depth-first search, Dijkstra's algorithm, Kruskal's algorithm, the nearest neighbour algorithm, and Prim's algorithm [3; 10]. However, searching in enormous

networks still remains a challenge with many of the interesting problems being NP-Hard. The NP-Hard problems include Clique, Independent set, Vertex cover, travelling salesman problem (TSP), Hamiltonian cycle, Graph partition, Vertex cover, Edge cover, and Graph isomorphism [2; 12]. In this paper we focus on TSP, and as a solution we suggest using meta-level agents over multiple agents in a network. The searching starts with a breadth-first approach and subsequently applies weighting costs for reducing the number of arcs to traverse.

Tools can be used for the route-finding problem in, e.g., computer networks, operations planning, and travel planning. Moreover, tools can be used for the touring problems, and visiting every city once (Hamiltonian path), or finding the shortest path for travelling salesman problem [3]. But they can also be used for the information overflow problem associated with Internet. Predicting optimal solutions in spatial networks remains a computationally hard problem. Although many attempts have been made, the problem of finding the paths through the large spatial space still remains. As a solution, we suggest calculating optimal decisions using multi-agents in networks. The optimal solution is held by a higher-level agent comprising the multi-agents involved in the solution and holds information about the computed travel time. These higher-level agents are called meta-level agents.

The characteristics of intelligent agents include operating under autonomous control, perceiving the environment, persisting over time, adapting to change and taking on another agent's goal [10, 14]. The agents, in multi-agent systems [15], can have these characteristics. The agents rely on prior knowledge with static characteristics, such as the nodes it connects and the distance. However, putting them in a real-world example will make them autonomous because the agents need to operate without inventions. The agents perceive their environment by considering the constraints and obstacles and subsequently act under the conditions that affect the agents' performance. To begin acting, the starting point for an agent is either data inserted by the user or a goal achieved by another agent. From this data or goal, the agents perform their task. At the lowest level, each agent performs one specific task and will unceasingly complete this task in the current system. However, if the constraints and obstacles in the task change, the agent needs to adjust to those changes. The agents continuously check the dynamic characteristics, and if needed, they adapt to their environment. Since optimal solutions usually include several agents, the system needs to take on other agents' goals. And as Roth [11] proposed, an agent can transfer commitments to another agent, our intelligent agents at the ground-level transfer environmental conditions in the network through the use of meta-level agents. The ground-level agents are information carriers between nodes in a network taking on the other agents' goals through the meta-level agents.

As software, we use a logic system for the ground-level multi-agents and meta-level agents. Each ground-level agent has knowledge about the route and moves by following the road, until it reaches its goal. Messages about time and the agents, involved in the optimal solution, are passed from the ground-level agents to the meta-level agents. From the messages, the meta-agents can collect the environmental conditions for every ground-level agent. By applying multi-agents in maps, the ground-level agents compute independently from other agents. With independency between the agents, we can use parallel computing and for maps we use simulated model of geographical information systems (GIS). The parallel computing is at the

agents' level where the geographical information systems keep the agents updated with environmental information.

2 Related work

The "Ant colony system" is a distributed algorithm that has been applied to the travelling salesman problem [4; 5]. The system uses a set of cooperating agents to find good solutions for the TSP. The agents cooperate using an indirect form of communication mediated by a pheromone deposited on the edges of a travelling salesman problem graph while building solutions [5]. Moreover, they use several agents per arc. In our work, we use agents in the arcs of the network, but we do not use the pheromone deposit on the edges. Instead we use information about the current circumstances. Take, for example, distance, constraints in speed due to topographic environment and temporary obstacles, such as current road conditions. The result of executing the agents is the time to accomplish their tasks, i.e., the travel time. Moreover, at each arc there is one agent working as an information carrier between the nodes. The basic principles of the "Ant colony" algorithm and its method of design and has been implemented in a multi-agent system [7]. Multi-agent structures can be described as an object or functional approach with an egalitarian subordination structure, a variable coupling strategy and an emergent constitution [6; 1]. The agents we use in our multi-agent system work locally in a spatial arrangement. Action of the agents is local modification. Moreover, the agents communicate locally and are perceptive of their local area via a GIS backend system.

Meta-reasoning for agents is when meta-level agents are controlling the agents' ability to trade off its resources between actions at an object level with those at a ground level. The meta-level control allows the agents to adapt their object-level computation. Moreover, the meta-level agents are horizontally modular [6]. For example, it has been shown that meta-level control with bounded computational overhead can allow complex agents to solve problems. The complex agents can solve problems more efficiently than current approaches in open dynamic multi-agent environments [9]. The meta-agents can be used to provide the control over agents and can stop the execution of some agents when further execution is not needed [9]. The meta-level agents collect data at a global level of knowledge, plan and schedule, and coordinate by using inter-agent negotiation. In our work, we do not apply meta-level control of the agents. Instead we use meta-agents to hold information collected by the agents at object level (ground level). The meta-agents are initiated from the work of the object-level agents and then used as a meta-level agent for the routes. The meta-level agents are then used to make leaps in the network ignoring some nodes without any loss of significant information. However, changes in the environment cause the meta-level agent to update in order cope with the new information as generated by the ground-level agents.

3 Multi-agents in the Network

The multi-agent system has several ground-level agents operating between nodes in a network. In the network, each agent acts as an information carrier and picks up all the information it apprehend during its execution, i.e., while moving from one node to another node. These two nodes are the only information the agent has from the beginning. During the execution, the agent works with the conditions in the environment. These conditions affect the agent's time for execution. This time is the most important feature, since this is what we use in the network and it affects the other agents.

In the network, an event will cause an agent to operate unless the agent has fulfilled the termination condition. Before executing the network, the agent does not know which node is the starting node and thus is symmetric. Which of the nodes becomes the starting node and the termination node for the particular agent depends on the user's input to the network or the other agents. The starting node either is the user input inserted to the system, i.e., the initial event, or an agent invoking another agent. The invocation occurs when an agent reaches its end node, which then becomes the start node for the agents associated with that node.

To collect information in the multi-agent system, the agents associated with the starting point execute the task of carrying information along the arc where the agents are working. These agents perceive their environment by knowing the starting and ending nodes together with the constraints in the arcs as well as obstacles. The constraints are permanent impacts in the environment and the obstacles are temporary problems, in which both constraints and obstacles become resistances to travelling in the arcs. Constraints are static information covering distance between nodes, allowed speed, topography and nature of the environment such as number of lanes. Obstacles are dynamic information about the conditions changing over a short period of time, like weather, road quality and field of vision but also degree of passable roads such as road constructions. The constraints and obstacles are not limited to these enumerated above and can be expanded with the information needed in the agents. Because of changes in the environment, the agents have to adapt to these changes while calculating the cost to move across the arc.

The time it takes to move between the nodes, distance multiplied with speed, is the initial cost for the agent. During the execution, the agents work under the circumstances input by constraints and obstacles, which affect the time it takes for the agent to travel. Each constraint and obstacle is weighted as an additional cost to the agent, and the degree of influence affects the size of the extra cost. Small influences give minor extra costs while greater influences increase the cost gradually. For the multi-agents in the network, each condition has a scale ranging from 0-5 multiplied with the factor of 2.5. This corresponds to reduce the speed from 90 km/h to 70 km/h. Worst-case condition makes the agents to increase their costs with 50, which reduces the speed with about 60%. This time is the key factor for calculating the optimal solution. After the execution, each agent keeps track of its travel time.

In the multi-agent system, each agent starts from a node, works through the arc, and ends up at a node. At this ending node, other agents are waiting for execution taking on the first agent's goal. As mentioned above, the result from the first agent is

the computed time of transit. Hence, the waiting agent cannot start executing before the first has reached the node with information. The information about the time will be carried further on in the network with the other agents. However, these agents also collect information about obstacles during their execution and the last agent round up the information from the agents operating from the starting node.

Since the multi-agents are working simultaneously from the starting node, they can work in parallel. As long as agents can work independently of each other, they will continue to execute until a termination node is reached by the system. However, reaching the termination node may not end the execution. If there are some agents still running at a lower cost than the agent that reached the terminating node, those agents will continue to compute until they either reach the terminating node or their costs exceeds the costs of the agent that had reached the final node.

A limitation of the multi-agents in the system is that the agents are not allowed to invoke other agents at a node already visited. Thus, if one agent already has been activated, this agent must not be called again for the same computation. Almost certainly, the agent is in a shorter path through the network. To know if the agent has been activated, each node has an execution flag, which holds the information about whether it has been visited by another agent.

When agents reach a node already visited, the agents that have the highest costs do not have any further possibility to execute and have to terminate. These can be removed from the calculation process without affecting the execution. The agents that run into dead-end roads also need to terminate directly. This speeds up the computational time because the system ignores these agents.

3.1 Example of a network with agents

To illustrate the multi-agents in a network, we provide an example of a graph symbolising a map with nodes and arcs. In the map, the nodes are cities and the arcs are the roads between the cities. In this example, the cities are denoted with the characters S and A-F where the S is the departure city and F is the destination city, see Figure 1. In the multi-agent system, these nodes have the real names of the cities. Between the cities, each road (arc) has an agent asserted to it.

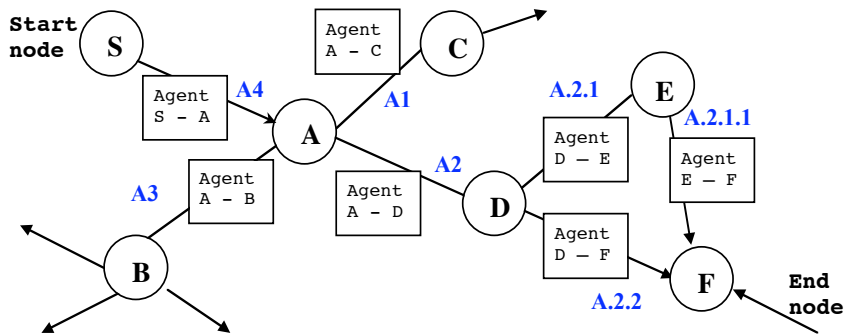


Fig. 1. An example of multi-agents in a network.

Each agent holds the information about the cities it works between, for example the agent S-A and agent A-D. In the figure, the roads also have character and number attached to distinguish them, e.g., A4, A2, and A2.1, corresponding to the real road denotations. These are used to collect and calculate the time for executing the agents. The agent S-A carries information between the start city (S) and the A city and agent A-D carries information between A and D city. The agent A-D and agent D-F carry information between the A and D and D and the end city (F), respectively. There is an alternative path through the network. It is the route from the starting city (S) to end city (F) through the A city, D city and E city. In this case, there are three agents involved instead of two agents. Although several agents are used, the route might be the optimal solution between the cities.

The information carried between the agents can be more or less relevant to the result of optimal solution in the graph optimisation problems. The shortest computation time to reach a goal is, of course, the optimal solution. However, it is interesting to know reason for selecting the particular route. Therefore, the agents need to keep information about the temporary conditions, such as road constructions, that probably affect computing time for the routes. There can also be more radical changes in the constraints, like changing the number of lanes. The amount of information to be kept can be decided for each computation through the network. However, a lot of information can affect the time for computation, which is the reason for introduce agents at a meta-level.

The system needs to keep track of the multiple agents in a network while finding the optimal solution because of its inherent heavy computing load. Therefore, we apply meta-level agents on top of ground-level agents, which preserve the result from executing the agents in the network. These meta-agents collect the other agents' computational results and save the results in order to finally present the optimal solution to the user. Also the meta-level agents can accumulate local information. That is, the agent can look for optimal paths between nodes in a local neighbourhood. This can be constructed as a side affect of other computations. When the meta-agents come through the agent, the agent can offer more interesting information to the meta-agent.

4 Agents at a meta-level

The meta-agents are created from ground-level agents after performing the calculation of the time it takes to execute the intended task. Hence, the meta-agents can be used to follow the multi-agents work through the network [8]. Meta-agents keep track of time and information about the conditions, passed as messages between the agents, i.e., ground-level and meta-level, in the network. Usually, several ground-level agents are involved in computing an assigned task. These agents' information, carried by each agent message, is used to build the meta-agent. The meta-agents execute in the context of the all the agents in the arcs. At the top-level, the meta-agent holds the goal of the computation. It will have the logic to determine when a solution has been achieved and to present the solution as the result of the computation. This logic is a set of conditions and constraints that select the best paths to help schedule the

computation and determine when a solution is reached. A simple approach to scheduling can be based on the shortest path generated at any time.

For the example mentioned above, the system produces one meta-agent. The meta-level agent comprises the ground-level agents involved in finding the optimal solution while moving from S to F, see Figure 2. The meta-agent is located at the top in the figure. This agent incorporates the information about the departure and destination and the time for executing each ground-level agent. The meta-agent also has the information about the nodes (cities) and conditions that the ground-level agents have collected during their execution, which is taken into account in the optimal solution. The ground-level agent to the left (agent S-A) has information about the initial start node, the road number and the stop node. After executing, the ground-level agents also have collected information about the constraints and obstacles, which have been translated into costs. The cost is sent as a message of time to the meta-level agent, (S 2.32). The ground-level agent to the right (agent E-F) has information about the activated node, road number and the end node.

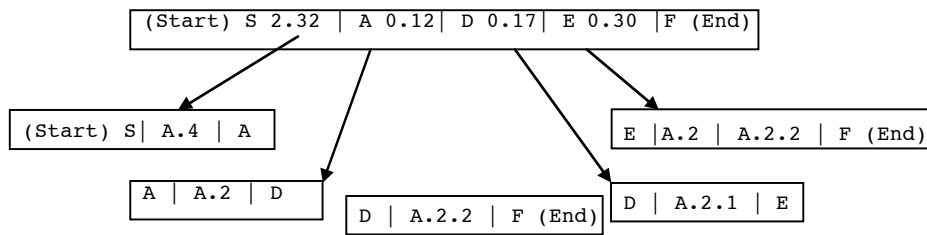


Fig. 2. Meta-level agents in the network

To the left in Figure 2, the first agent (agent S-A) is activated by the user. This agent reach the A node which causes the agent A-D, agent D-E, agent D-F and agent E-F to start running. The agent D-F is not captured in meta-level agent because it required higher cost than the other path, D-E and E-F. Additionally, for each agent, the cost for executing between the arcs is hold by the meta-agent. The cost for S-A agent is 2.32 hour, A-D agent is 12 min, D-E agent 17 min and E-F agent 30 min.

The selection of the optimal solution is made by the meta-agents. The meta-agent enters each node of corresponding agents in the network and performs the calculation for the agents. It has access to all information contained in the ground-level agents, including all collected derived conclusions. The meta-agent simply adds the agents' performance in time, taken into account the static information and the dynamic information.

It is possible to search for multiple solutions in the network. For example, a practical problem is to set up a set of routes for multiple deliveries to nodes in the network. This is a useful problem for airlines, trucking and other carriers. In this case, several meta-agents can be started simultaneously to search the network for solution. For each computation in parallel, several meta-agents may be created since the first agent to reach the end node might not be the fastest. Then, the meta-agents must be tested against each other.

5 Software for the agents in the network

As software, we use a logic system for the ground-level agents and meta-level agents. The network is a small map of a part of Sweden and part of United States of America. Between the nodes there are several arcs, which are corresponding to the reality. Each ground-level agent has knowledge about the route and follows the road, until it reaches its goal. Messages about time are passed from the ground-level agents to the meta-level agents, which are stored in the database of the system. The meta-level agent can keep track of the dead end roads and make the ground-level agent to avoid running in to it again.

For applying multi-agents in maps, we use parallel computing in simulated geographical information systems. The parallel computing is at the ground-agents' level. To fully utilise the parallel facility, we duplicate the agents for each node that has two or more computing agents. The number of computing agents decides the number of duplicates. In this way, the system can continue to execute all the agents without needing to explore one at the time.

By applying meta-level agents, we can calculate the optimal path for between two nodes. This facility is used for calculating the optimal path through the graph, producing several meta-agents and comparing the cost for the meta-agents. The current system gives alternative paths between nodes, especially when the agents are visiting several cities on their way to the goal. The costs are used as a weighting facility when choosing the optimal road.

6 Conclusions and further work

In this paper, we have presented an approach of using meta-level agents for multi-agents in networks to suggest a solution to the optimal solution problem in graphs. The intelligent agents follow the arcs and compute the time by acting upon the arc information, which are the constraints and obstacles that the specific arc has about the environment. The computation time becomes the significant factor to find the optimal solution. To keep track of the computational time of the ground-level multi-agents, the network also uses meta-level agents to which the ground-level agents pass the computed time as messages. Beside time, the meta-level agents can also hold environmental information from the ground-level agents. The use of meta-level agents for multi-agents in networks support parallel computing of independent ground-level agents. The parallel agents work as long as ground-level agents are independent of other agents and have not reached a node that is already reached by another agent. However, when the agent has been activated from a node, this node turned the execution flag to "visited". This facility needs a lot of testing before deciding that it works in all situations.

Moreover, to apply multi-agent in reality, we need to develop a parallel computing system using a GIS interface. The parallel computing handles the multi-agents, and the geographical information systems provide the agents with updated environmental information. The system also needs testing to check the extent to which the GIS interface supports the agents.

One appealing aspect of meta-agent approach is the ease of scheduling the multi-agents for execution. One simple approach is to partition the agents onto processors. The agents on a processor remain idle until something affects them and causes them to run. This can work in a loosely coupled parallel processing system. A second approach is to have a single scheduling queue on a central processor that can assign work to processors as available. Clearly, there are trade-offs involved, especially with the amount of information to be moved between processors. Also, we have the question of whether or not the work is evenly distributed between agents or whether there are some agents that are executed more frequently than others.

References

1. Attoui, A: *Real-Time and Multi-Agent Systems*, 1th edition, Springer ISBN: 1-85233-252-2 (2000)
2. Berman, K. and Paul, J.: *Algorithms: Sequential, Parallel, and Distributed. Course Technology*; 1th edition, ISBN-10: 0534420575 (2004)
3. Cormen, T. Leiserson, C. Rivest, L. and Stein, C.: *Introduction to Algorithms*. Second Edition. MIT Press and McGraw-Hill, 2001. ISBN 0-262-03293-7 (2001)
4. Dorigo, M. Maniezzo, V. and Colorni, A.: Ant System: Optimization by a Colony of Cooperating Agents. *IEEE Transactions on Systems, Man, and Cybernetics*, Part B, 26 (1): (1996) pp. 29–41
5. Dorigo, M., and Gambardella, L.M.: Ant colony system: a cooperative learning approach to the traveling salesman problem. *IEEE Transactions on Evolutionary Computation*, ISSN: 1089-778X Apr 1997, Volume: 1, Issue: 1. (1997) pp. 53-66
6. Ferber, J.: *Multi-Agent Systems* Addison Wesley Co. ISBN 0-201-36048-9 (2002)
7. He, J-m., Min, R. and Wang, Y-y.: Implementation of Ant Colony Algorithm Based-On Multi-agent System. ICCNMC 2005 (2005) pp. 1234-1242
8. Håkansson, A., and Hartung, R. L. Using Meta-Agents for Multi-Agents in Networks. (ICAI'07) *The 2007 International Conference on Artificial Intelligence, WORLDCOMP'07*, Las Vegas, USA, June 25th -28th, (2007)
9. Raja, A. and Lesser, V.: A Framework for Meta-level Control in Multi-Agent Systems. *Autonomous Agents and Multi-Agent Systems*, Springer Online. January (2007)
10. Russell, S., and Norvig, P.: *Artificial Intelligence - a Modern Approach*, Prentice Hall (2003) pp. 32-752.
11. Roth, V.: Mutual protection of co--operating agents, in Vitek J. and Jensen C., editors, *Secure Internet Programming: Security Issues for Mobile and Distributed Objects*, LNCS 1603, New York, NY, USA: Springer-Verlag, (1999) pp. 275-285.
12. Skiena, S.: *The Algorithm Design Manual*. Springer; 1 edition, ISBN-10: 0387948600 (1998)
13. Turban, E., Aronson, J. and Ting-Peng Liang: *Decision Support Systems and Intelligent Systems*. 7th edition. Pearson Prentice Hall. ISBN: 0-13-123013-1 (2005)
14. Wooldridge M. and Jennings N. *Intelligent agents: Theory and practice*. Knowledge Engineering Review 10(2), (1995)
15. Wooldridge, M. *An Introduction to MultiAgent Systems*, John Wiley & Sons Ltd, ISBN 0-471-49691-X. (2002)