Using Pheromones, Broadcasting and Negotiation for Agent Gathering Tasks

Simon Coffey and Dorian Gaertner

Imperial College London, SW7 2AZ, United Kingdom

Abstract. We describe an implementation of distributed, multi-threaded BDI-style [RG95] agents cooperating efficiently in a food-collecting scenario. Using ant-style pheromone trails and a pseudo-random walk procedure they explore the world uniformly, and negotiate to allocate collection tasks. Global information is disseminated via a publish/subscribe mechanism. The system is implemented using the concurrent logic programming language Qu-Prolog.

1 Problem description

For brevity's sake, a full description of the problem is omitted here. However, in addition to the constraints given in the competition specification, we have made two further assumptions: that the agents can move only in the directions North, South, East and West (i.e. diagonal moves are excluded), and that each agent can only carry one item of food at a time.

2 Design

A multi-agent system is typically characterised by the *distributed execution* of *communicative agents* that are *situated* in an environment.

We decided to use multi-threaded, logic-based, autonomous pseudo-BDI agents that are situated in an environment without central control. The environment process seeds food into the world, maintains the pheromone trails[1], sends percepts to the agents when requested and interfaces with the GUI.

2.1 Architecture

We designed our agents using an architecture loosely based on Rao and Georgeff's Beliefs-Desires-Intentions (BDI) model [RG95]. Each agent has *beliefs* about the state of the world including the location of food and the depot as well as beliefs about *claims* other agents have made. When an agent claims a certain piece of food, he informs the other agents about his *intention* to pick it up. Delivery of food and searching are other examples of intentions. *Desires* in our implementation are largely implicit, being limited to the built-in aim of each agent to collect and deliver as much food as possible in the shortest number of moves.

Each agent consists of two primary threads and a dynamic database (figure 2(a)). The knowledge thread receives percepts from the environment (*sensing*), updates the belief store depending on how it perceived the world, re-evaluates the intentions of the agent and communicates with other agents announcing certain events. The action selection thread then uses the current beliefs and intentions to decide which action to execute next. It informs the environment about its choice of action, which updates the world state and sends new percepts to the agent's knowledge thread.

[1] described in Section 2.8

2.2 Agent Language

Actions, percepts, beliefs and intentions are all sets of Prolog terms:

```
Action ::= [pickup, putout, move(Direction)]
```

where Direction is a variable representing north, south, east or west

```
Percept ::= [depot_same_cell, food_same_cell, has_food, has_moved,
             north(N), south(S), east(E), west(W)]
```

where N, S, E, and W are variables that represent cart, wall or a pheromone level

```
Belief ::= [at(X,Y), depot_at(X,Y), have_food, intends(Agent,
            Intention)]

Intention ::= [collect_food(X,Y), deliver_food]
```

where X and Y represent coordinates. Note that searching is never explicitly intended by the agent, but used as a default behaviour if no other intentions exist.

2.3 Action Selection

Agents choose their actions using *teleo-reactive (TR) programs* [Nil94], consisting of a priority-ordered sequence of condition/action rules. A simplified version of the TR program used is shown in Figure 1. This approach is particularly useful in scenarios like ours, in which durative behaviours (e.g. *explore*) are desired. It is important to note that at each percept/reaction cycle, the action chosen is only ever a single atomic one, belonging to the agent's set of allowed actions (defined above). For example, while `walk_to(X,Y)` appears to be a multi-step plan, it is in fact simply a set of rules which choose the agent's next atomic action; it must be repeatedly invoked in order to arrive at `(X,Y)`. Thus, the right-hand side of the rules in figure 1 are all either atomic actions, or programs which return an atomic action.

$$
\begin{aligned}
\texttt{intends(deliver\_food)} \wedge \texttt{believes(agent\_at(depot))} &\rightarrow \texttt{putout} \\
\texttt{intends(deliver\_food)} \wedge \texttt{believes(depot\_at(X,Y))} &\rightarrow \texttt{walk\_to(X,Y)} \\
\texttt{intends(deliver\_food)} &\rightarrow \texttt{explore} \\
\texttt{believes(at\_food)} &\rightarrow \texttt{pickup} \\
\texttt{intends(collect\_food(X,Y))} &\rightarrow \texttt{walk\_to(X,Y)} \\
\top &\rightarrow \texttt{explore}
\end{aligned}
$$

Fig. 1. Simplified action-selection TR program

Note that the action selection program does not manipulate beliefs, alter the intentions of the agent or handle negotiation in any sense; it operates solely on the current intentions and beliefs of the agent, returning only an action. All agent state manipulation is performed by the intentions thread (described in Section 2.4), which runs in parallel to the action thread, ensuring a consistent set of beliefs and intentions for the action selection program to use.

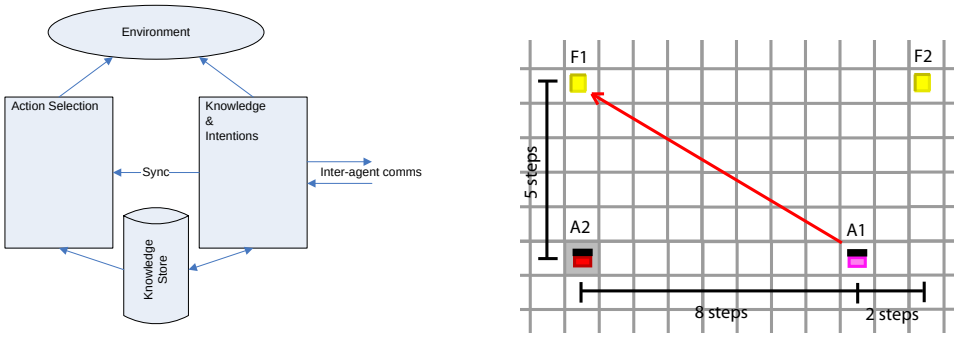
2.4 Intention Selection & Knowledge Maintenance

The intention selection thread takes the form of a message-processing cycle. While awaiting the next set of percepts from the environment, it listens for broadcast messages and negotiation requests from other agents, updating its beliefs and intentions accordingly. This is the only place in which modification of the agent's `believes(...)` and `intends(...)` dynamic predicates is permitted. For example, if agent `red` receives a broadcast message informing him that agent `blue` is claiming food at location (5,9), it will add the term `believes(intends(blue, collect_food(5,9)))` to its dynamic knowledge base.

When a set of percepts is received, the agent first updates its beliefs about the world state using the new percepts. Since the set of percepts it can receive is relatively limited, this is achieved with an explicit set of handling routines for each type of percept. It then decides whether to send any negotiation requests, and finally re-evaluates its intentions accordingly. It does so using a series of declarative conditions, made possible by the backtracking operation of Prolog-style languages. For example, the delivery cost function for a particular item of food is simply written with two rules,

```
cost_of(food(X,Y),Cost) :-
    believes(agent_at(AgX,AgY)),
    believes(depot_at(DepX,DepY)),
    manhattan(AgX,AgY,X,Y,C1),
    manhattan(X,Y,DepX,DepY,C2),
    Cost is C1 + C2.
cost_of(food(X,Y),Cost) :-
    believes(agent_at(AgX,AgY)),
    manhattan(AgX,AgY,X,Y,Cost).
```

where `manhattan(X1,Y1,X2,Y2,D)` gives the manhattan distance between two points. This cost function is then called to find the optimum choice of food at the start of each turn (assuming there is any known food). If this food is believed to be claimed by another agent, negotiations are initiated with that agent. If the negotiation is unsuccessful, the agent will claim the cheapest unclaimed food (or retain whichever food it had previously claimed). Every new claim is broadcast to the other agents, enabling them to contact the "owner" of any food they wish to claim for themselves.



(a) (b)

Fig. 2. (a) Negotiation example and (b) architecture design

2.5 Communication

Communication between agents utilises two of the main communication paradigms: publish/subscribe, and point-to-point messaging. The former is used for global knowledge sharing, while the latter is used for efficient negotiation between specific agents, as well as agent/environment communication. Each agent places a *subscription* for messages about food/depot locations and agent commitments at a remote server. When one agent finds an item, or claims some known food, it will *publish* a notification about this event to the server which in turn will inform all other agents that have subscribed to this event. This allows for dynamic addition of new agents to the scenario without having to change the running system. Negotiation is achieved using asynchronous message-passing. This is more efficient than using the broadcast system, since negotiation is always bilateral in our implementation; there is no need for all agents to be party to the negotiation messages.

2.6 Negotiation

In order to most efficiently allocate the collection of known food to each agent, we allow our agents to negotiate over the targets of their intentions. The agents have a defined policy only with respect to individual negotiations, namely to minimise the combined cost of delivery for the two negotiating agents. This is achieved by examining each agent's next-best option, and optimising accordingly. The implicit global effect of this policy is to minimise the total delivery cost of all known deliveries. This is the result of a series of bilateral negotiations; no single agent takes responsibility for optimising the entire set of deliveries.

Figure 2(b) illustrates an example where negotiation can improve the efficiency of food collection. It shows a snapshot of the environment state, in which agent A_2 has just delivered some food, agent A_1 has claimed and intends to pick up food F_1 and some other agent that was already carrying food accidentally discovered and broadcasted the existence of food F_2. Without negotiation, A_2 would claim F_2 and collect and deliver it in 30 steps while A_1 drops off F_1 in 18 steps.

Note that A_1 would not volunteer to pick up F_2 since this would increase his personal delivery cost to 22 steps. We therefore allow A_2 to send a bid to A_1, requesting permission to collect F_1 instead. It sends its cost of collecting F_1, plus the delivery cost of its next-best option (in this case F_2). A_1 will then consider the request, ceding responsibility for F_1 if the total delivery cost after the swap is reduced. The re-allocation allows agent A_2 to pick up and deliver F_1 in 10 steps at the expense of a small increase in the other agent's delivery cost. In terms of welfare economics, both the egalitarian and the utilitarian social welfare is improved[2].

2.7 Agent Roles

After initial experimentation with all agents performing as described above (i.e. searching until they first find food, then immediately delivering it), it became obvious that except in the most food-rich environments, knowledge about food locations was almost non-existent. The agents thus simply randomly walk until they first find food, which they immediately pick up and deliver, giving no opportunity for task optimisation. We therefore implemented a second type of agent, a scout. Upon finding food, a scout will not pick it up, but will merely broadcast its location to the gatherer agents, and continue searching. This can be viewed as a second implicit desire, with the scout agent's desire being to gather information rather than food.

[2] We take utility to be the negated cost of delivery, so that a shorter delivery yields a higher utility. The cost of delivery is the number of steps from the agent's location via the location of the food to the depot location.

As implemented, the scout is statically determined; it may not switch to delivery mode. However, it is easy to envisage a scheme in which agents switch to scouting dynamically, thus completing the full BDI repertoire of mental attitudes. In a scenario in which all the agents are randomly searching, the first agent to happen upon food might switch to scout duty, combing the rest of the area while the other agents collect the food it has discovered. In this manner, unexplored areas (and hence concentrations of food) would be explored, rather than the first food simply being delivered and forgotten about.

2.8 Exploration

Initially the agents do not know where the depot is located or which cells of the grid contain food. They must therefore explore the world around them. Dividing the world into quadrants and assigning each agent to a quadrant would be the most efficient way to explore the world completely, but this ignores over-exploration of repeatedly visited areas (i.e. the area around the depot). For this reason, a pseudo-random walk technique is used for exploration, utilising trail markers to ensure that agents prefer to explore cells that have not been as frequently visited.

A completely random walk based on Brownian motion would not be efficient enough since it tends to over-explore some areas at the expense of others. We chose to implement a more directed approach based on pheromones. Each agent drops a fixed amount of pheromone each time it enters a cell, similar to the methods used in ant colony optimisation ([DMC96], [GC05]). An agent can *smell* the concentration of pheromone in its neighbouring cells and probabilistically decides to move in a direction which is under-explored. If there are one or more unexplored adjacent cells, it will always choose a move to one of these cells. This pseudo-random walking leads to the uniform exploration of the world. It also compensates for over-exploration of the area around the depot; the repeated trips of agents to this location mean it would be heavily over-explored if a quadrant-based strategy were used.

However, there is a disadvantage to simply counting all the visits to a cell since the start of the simulation. In an environment in which food is continuously appearing, the goal of a search algorithm must be to ensure each cell is visited as regularly as possible. In a system with permanent pheromones, a cell that has been visited only once, but very recently, appears more attractive to explore than a cell that was visited 10 times, 100 turns ago. In fact, the opposite is true, and the cell with "stale" pheromones should be explored preferentially. For this reason, a pheromone decay mechanism has been implemented and proved useful, whereby pheromone values decrease over time according to a variety of formulae. This ensures that cells which were over-explored in the past do not get unreasonably ignored.

A further advantage of random walking is its use in avoidance of deadlocks. When two agents block each other's paths they will randomly move out of the way. While they may not successfully avoid each other instantly, due to the random choice of direction, the avoidance routine inevitably resolves the deadlock, since it is statistically impossible for both agents to choose the same move for ever. In addition, this method is much simpler to implement than exhaustive characterisation of every possible deadlock, along with explicit strategies for resolving them.

Most importantly, using this flexible movement behaviour our implementation adapts very easily to unknown and even dynamically changing environments.

3 Implementation

Our design requires an implementation language that allows for multi-threaded execution of agents. We chose *Qu-Prolog* [RW03] because it allows for easy, declarative description of the higher-level reasoning involved in intention selection and

negotiation. Its flexible system of dynamic predicate manipulation also provides an unconstricted environment in which to construct and modify the simple agent language we have used, while simultaneously being descriptive enough to allow the lower level algorithms to be concisely expressed.

The publish/subscribe mechanism we described in Section 2.5 is realised using broadcasting via an *Elvin* [SAB+00] server. Direct negotiation between agents makes use of the Interagent Communication Model (ICM). Its communication server provides agent naming facilities and the means to encode, transport and queue symbolic messages.

Effectively, we are using three forms of communication—point-to-point communication for negotiations, broadcasting for events and knowledge sharing, and indirect communication via the environment using pheromones for exploration.

4 Analysis/Conclusion

The foundation of the broadcast and negotiation techniques implemented is a good supply of information about the environment. In scenarios where there is more known food than the agents can collect at once, these techniques have a potential to improve the utilisation of the agents, since the time spent conducting relatively unguided searches is limited. However, this knowledge of food locations needs first to be obtained, hence the introduction of a scout agent. The impact of the various techniques implemented is briefly assessed here.

For our quantitative analysis, we fixed the depot at location (10,10) and ran the simulation until 100 items of food had been collected and delivered. The agents were still required to discover the depot on each run. Ten runs of the simulation were conducted for each scenario. The average number of steps for one food delivery (μ_{steps}), the standard deviation of the number of steps (σ_{steps}) and the average number of successful negotiations per delivered food item (μ_{neg}) have been measured.

Table 1. Quantitative results when food is seeded every 20 seconds

Scenario	μ_{steps}	σ_{steps}	μ_{neg}
4 gatherers (no pheromone, no negotiation)	73.7	3.32	n/a
4 gatherers (pheromones, but no negotiation)	55.7	1.27	n/a
4 gatherers (pheromones and negotiation)	56.4	2.07	0.20
3 gatherers and 1 scout (pher. and neg.)	55.6	1.1	0.29

This shows that adding guidance to the randomly walking ants with the help of pheromones significantly improves their behaviour. However, adding negotiation does not seem to improve the results. We believe this is due to the low rate at which food is seeded into the environment. The negotiation usually improves a combined delivery of two ants by about 10%. However, the ants spent the majority of their time searching for food or the depot. Only about 10% of their time is spent collecting and delivering and so the improvement achieved by adding negotiation is only 10% of 10%, equivalent to 1% overall.

In the high seeding-rate environment (with food seeded every 7 seconds), having a dedicated scout agent proves significantly detrimental to the team performance. This is unsurprising, as with high rates of seeding, food is sufficiently abundant that the agents have no trouble finding food on their random walk. The team with a scout still performed better than would be expected of a team consisting solely of three gatherers, however, experiencing only a 9% performance drop despite effectively losing a quarter of the delivery capability.

Table 2. Scout effect with varying seeding rates

Scenario	Seeding	μ_{steps}	σ_{steps}	μ_{neg}
4 gatherers	high	28.9	0.74	0.22
3 gatherers and 1 scout	high	31.6	1.28	0.18
4 gatherers	low	56.4	2.07	0.20
3 gatherers and 1 scout	low	55.6	1.1	0.29

In the low seeding-rate environment (with food seeded every 20 seconds), however, the benefit of the scout agent completely compensated for the loss of delivery capacity, roughly equalling the delivery rate of four gatherers. In effect, the scout provides sufficient global knowledge to allow the agents to employ their high-level reasoning much more frequently, resulting in more efficient collection of the known food. As the delivery agents leave trails every time they visit the depot, the scout agent tends to explore the areas further from the depot, discovering concentrations of food that the delivery agents are unlikely to find.

As predicted, the usefulness of the scout agent inevitably depends on the rate of seeding. In an environment with a high rate of seeding, the food density is such that it becomes more efficient to simply have all agents collecting, since they are likely to find food soon after leaving the depot on their random search. Additionally, a scout agent in this situation tends to over-explore the edges, drawing the gatherer agents further from the depot than necessary. In a food-sparse (and thus information-poor) environment, however, the scout becomes more useful despite the loss of one agent's delivery capacity.

While our preliminary results do not show a scenario in which a scout agent provides a decisive advantage, they do indicate that there are situations in which a definite benefit exists. As a future avenue of investigation, we believe that allowing the agents to dynamically switch roles offers potentially superior results.

5 Acknowledgements

We would like to express our deepest gratitude to Silvana Zappacosta for allowing us to use and helping us to modify her visualisation written in Tcl/Tk. Furthermore, we are indebted to Keith Clark for encouraging us to participate in this contest and for many fruitful discussions.

References

[DMC96] Marco Dorigo, Vittorio Maniezzo, and Alberto Colorni. The Ant System: Optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics Part B: Cybernetics*, 26(1):29–41, 1996.

[GC05] Dorian Gaertner and Keith Clark. On optimal parameters for ant colony optimization algorithms. *Proceedings of the International Conference on Artificial Intelligence 2005 (to appear)*, 2005.

[Nil94] Nils J. Nilsson. Teleo-reactive programs for agent control. *Journal of Artificial Intelligence Research*, 1:139–158, 1994.

[RG95] A. Rao and M. Georgeff. BDI agents: From theory to practice. *Proceedings of the 1st International Conference on Multi-Agents Systems*, pages 312–319, 1995.

[RW03] Peter J. Robinson and Michael J. Walters. *Qu-Prolog 6.3 Reference Manual*. The University of Queensland, 2003.

[SAB+00] B. Segall, D. Arnold, J. Boot, M. Henderson, and T. Phelps. Content based routing with Elvin4. In *Proceedings of AUUG2K*, June 2000.