Abstract- An approach to evolving globally coordinated behaviours in groups of autonomous mobile robots is presented. The control system in each robot is identical and consists of a cellular automaton which serves to arbitrate between a number of fixed basis behaviours. Genetic algorithms search for cellular automata whose arbitration results in success on a predefined task. Heap formation is presented as an example of a task requiring global coordination. Simulation results are provided.

1 Introduction

Researchers in collective robotics often draw attention to social insects to explain the motivation behind their work (Kube and Zhang, 1996; Dagaeff et al., 1997; Deneubourg et al., 1990; Theraulaz et al., 1990). In the absence of a central controlling agent, colonies of ants are able to work together to the benefit of the society as a whole (Hoyt, 1996). Each ant behaves according to its local situation yet interesting and coherent global behaviours result. No particular ant is essential to the overall dynamics. Surely there must be some way in which we can harness the principles driving these insects to work together and apply them to artificial systems. In a basic sense, collective robotics has to do with designing control methods for populations. It is hoped that the benefits of such work indeed may be twofold in that it may guide the engineer in the design of distributed systems while at once furthering our knowledge of how biological collectives achieve coordination.

There are many possible approaches available to the would-be collective roboticist. Taking inspiration from biological systems does not compel us to mimic them cell for cell. As (Pfeifer, 1996a) puts it,

The great advantage of the synthetic approach is, of course, that we have built the agents ourselves, i.e., we know what is in our systems, and that we can experiment with alternatives as much as we like.

A major first consideration in the design of a population controller, is whether the method should be centralized or decentralized. Centralized control refers to the presence of a central controlling agent which dictates what all other agents are doing at all times. This type of method might be easier to implement initially but it can encounter serious scalability problems. Scaling refers to increasing/decreasing the number of agents in the population. Centralized control is really no different than the control of a very complex single agent. As the population becomes larger, that single agent becomes increasingly complicated and difficult to control.

At the other end of the spectrum is decentralized control. Each agent in the population behaves according to its own set of rules; it is autonomous. These rules allow the agent to communicate with a number of other agents in the system. Explicit communication can either be sparse (usually local) or global. Implicit communication (sometimes called stigmergy\(^1\)) is always local as it means agents interact only through physical constraints (e.g. two agents colliding). If all the agents behave according to the same rules the population is said to be homogeneous; otherwise it is heterogeneous. Decentralized control is typically very scalable if local communication is used (the bandwidth of communication for each agent is fixed). However, finding communication rules which produce useful behaviours is not always (or even sometimes) easy.

The choice of centralized versus decentralized control is a question of philosophy. For the engineer working on a very specific problem with a small, fixed number of agents, centralized control might be the easy answer. For the scientist, interested in learning the general principles involved in multiagent or self-organizing systems, decentralized control is the more elegant approach. In fact, learning how systems in nature are able to self-organize is perhaps one of the most important problems of the day. How do many simple units interact to produce something that is more than the sum of the parts? What, if any, are the general principles of self-organization? How can we design artificial systems which are able to harness this power which is so prevalent in nature? How do those little ants work so well together to produce such a coherent global behaviour? In case there is any doubt, the approach to be presented here will be decentralized. Furthermore, the collective will be homogeneous and will use implicit local communication.

The flow of this paper will be as follows. The basic architecture of a single agent will be given first; this is the part of the system that does not evolve. The part of the system

\(^1\)The term was coined by Pierre-Paul Grassé in the 1950s to mean ‘indirect communication taking place among individuals in insect societies’. Grassé studied termite nest formations.
that is evolved, based on a cellular automaton, will then be explained, followed by a description of the evolutionary process, a genetic algorithm. A task requiring global coordination (heap formation) will be presented followed by some results of the method in a software simulation. The last section will deal with the scalability of the solution.

2 The Single Agent

Although this is an attempt to lay out a general approach to collective robotics, some basic assumptions will be made about the nature of the individual robots. Each robot should have a number of sensors and actuators. It will be assumed that some transformation may be performed on the raw sensor data to achieve a set of independent (sometimes called orthogonal (Kube and Zhang, 1996)) virtual sensors which output a discrete value. (Kube and Zhang, 1996) divides orthogonality into two types: spatial and modal. Spatially orthogonal sensors do not look at the same region of space. Modally orthogonal sensors do not look at the same type of data (imagine two sensors, one which can only detect balls and another which can only detect obstacles). Figure 1 provides a visualization. The transformation is essentially a preprocessing step which reduces the raw sensor data to more readily usable discretized inputs.

![Figure 1: A robot with an orthogonal sensor array.](image)

Let us further assume that the output of our control system may be discrete. This may be done by way of a set of basis behaviours (Matarić, 1997). Rather than specify the actuator positions (or velocities), we assume that we may select a simple behaviour from a finite predefined palette. This may be considered a postprocessing step which takes in a discrete input sequence (size \( N \)) and outputs the appropriate discrete output (one of \( M \) basis behaviours). There are several candidates for this role but the one affording the most general decision surfaces between input and output is a straightforward lookup-table similar to cellular automata (CA). In fact, this work follows on that of the Evolving Cellular Automata (EvCA) group at the Santa Fe Institute, New Mexico. In various papers (Das et al., 1995; Mitchell et al., 1993; Mitchell et al., 1996; Hordijk et al., 1998) this group showed that genetic algorithms were able to evolve cellular automata which performed prescribed tasks requiring global coordination. This is essentially what we wish to achieve but we have the added difficulty of dealing with the physical environment of our robots.

This type of lookup-table control in autonomous robots is often called reactive. For every possible input sequence the CA scheme stores a discrete output value. In other words, for every possible input view there is an output corresponding to one of the basis behaviours. At each time-step, the robot looks up the action which corresponds to its current view and carries it out. The size of the lookup-table (for binary sensors) will be \( 2^N \) so if there are too many input sensors the CA lookup-table will be very large; the approach is therefore usually rendered feasible for only modest numbers. Although the number of basis behaviours \( M \) does not directly affect the size of the CA lookup-table, it does affect the number of all such possible tables which is \( M^{(2^N)} \) for binary sensors. Again, modest numbers of basis behaviours keep the size of the search space reasonable.

3 Genetic Algorithms

The crucial step in this whole approach is the discovery of particular CA lookup-tables which cause a collection of identical robots to succeed on a task requiring global coordination. In fact, the whole field of collective robotics could be described as finding a set of local rules which produce the desired global behaviour, if they indeed exist, and then finding out why those rules stand out. The obvious first method to attempt is to design the local rules by hand. John Horton
Conway picked his Game of Life rules seemingly out of a hat with much success. How hard can it be? Well, it turns out to be very difficult to do this for all but the most trivial examples. Even when working with simple one-dimensional cellular automata models (a far cry from a fleet of robots) there can be millions if not billions of sets of local rules from which to choose. We are typically faced with a combinatorial explosion. Typically one of two methods is used in finding the local rules which produce interesting global behaviours: optimization or reinforcement learning. In the spirit of the EvCA group, an evolutionary global optimization technique will be employed, namely a genetic algorithm (GA).

GAs are based on biological evolution. For a good review see (Goldberg, 1989) or (Mitchell and Forrest, 1995). A random initial population of \( P \) CA lookup-tables is evolved over \( G \) generations. Each CA lookup-table, \( \phi \), has a chromosome which consists of a sequence of all the discrete values taken from the table. At each generation, a fitness is assigned to each CA lookup-table (based on how well a collection of robots, each containing the CA lookup-table, conforms to our prescribed behaviour). A CA lookup-table’s fitness determines its representation in the next generation. Genetic crossovers and mutations introduce new CA lookup-tables into the population. The best \( K \leq P \) CA lookup-tables are copied exactly from one generation to the next. The remaining \( (P-K) \) CA lookup-tables are subjected to a single site crossover at a random location with probability, \( p_c \). Furthermore, they are subjected to random site mutations with probability, \( p_m \), per site. Both \( p_c \) and \( p_m \) are selected from a uniform random distribution.

4 Heap Formation, a Sample Task

Sometimes called the shepherding task, heap formation has an established history in the literature and is directly comparable to the behaviour in some insect societies (Maris and te Boekhorst, 1996; Dagaeff et al., 1997; Deneubourg et al., 1990). It is believed that this task requires global coordination for a group of agents, existing in a two-dimensional space, to move some initially randomly distributed objects into a single common location. However, there is no central controlling agent which says where to start the heap. The agents must come to a common decision among themselves without any external help (analogous to the global partitioning task of the EvCA group (Mitchell et al., 1996)). The absence of any central controlling agent or goal beacon makes this a difficult computation to be performed by this spatially extended system.

Traditional genetic algorithms require a fitness function to be defined (on which selection is based). For the heap formation problem, the physical space in which the robots exist is broken into \( J \) cells, \( A_j \), and the fitness function defined to be

\[
 f_{total} = \frac{\sum_{i=1}^{I} f_i}{I} \tag{1}
\]

where \( I \) is the number of random initial conditions over which \( f_i \) is averaged; \( f_i \) is the fitness on one initial condition only, given by

\[
 f_i = 1.0 + \frac{\sum_{j=1}^{J} p_j \ln p_j}{\ln J} \tag{2}
\]

where \( p_j = n(A_j)/\sum_{j=1}^{J} n(A_j) \) and \( n(A_j) \) is the number of objects in cell, \( A_j \). This is essentially a “Shannon entropy” type number only it is 0 when the objects are equally distributed over all cells, and 1 when all the objects are in a single cell. To summarize, fitness is assigned to a CA lookup-table by equipping each robot in a collective with that CA lookup-table. The collective is allowed to roam around in a two-dimensional space which has a random initial distribution of objects. At the end of \( T \) time-steps, \( f_i \) is calculated, which indicates how well the objects are heaped. This is all repeated \( I \) times to allow some statistical averaging and \( f_{total} \), the fitness of the CA lookup-table, is determined.

5 Simulation Results

![Figure 3: Typical view of a software agent. Each agent sees only 6 squares (local information). Darker circles (without lines) are objects. The lighter circle (with a line indicating orientation) is the agent itself.](image)

Up to this point everything proposed could have been applied to a population of real robots. However, from this point on we will describe a particular implementation of the method in a simulation. For simplicity, the space in which the agents roam will be a two-dimensional lattice with square cells and periodic boundary conditions in both spatial directions (the surface of a torus). Each agent will be able to see only 6 squares as depicted in Figure 3. Each agent is able to carry one object at a time and can tell whether or not it possesses an object. In five of the squares an agent sees there are 3 possibilities (nothing, object, agent); in the sixth square, the one occupied by the agent, there are 2 possibilities (carrying object, not carrying object). This means there are \( 3^5 \times 2 = 486 \) entries in the CA lookup-table. Only 2 basis behaviours will be defined for this task.

- **Move Self** in which the agent moves forward if the cell in front is empty, otherwise turns left if the cell to the left is empty, otherwise turns right.
- **Manipulate Object** in which the agent picks up/puts down an object from directly ahead (if possible); other-
wise picks up/puts down an object left of center (if possible), otherwise picks up/puts down an object right of center (if possible), otherwise activates the Move Self basis behaviour. Whether the agent is picking up or putting down an object depends on whether it already has one in its possession or not.

The choice of these behaviour modules is arbitrary yet natural. With 2 basis behaviours and a CA lookup-table of size $2^{486} \approx 10^{146}$ possible CA lookup-tables. This number is quite large but, as we will see, good solutions can still be found. It should be pointed out that our agents will be functioning in a completely deterministic manner. From a particular initial condition, the system will always unfold in the same particular way.

In our experiments, we have used a GA population size of $P = 50$, number of generations $G = 150$, keepsize $K = 5$, crossover probability $p_c = 0.6$, and mutation probability $p_m = 0.005$. For the purposes of finding good rules we have used a two-dimensional world of size $31 \times 30$, 30 agents, 60 objects, a training time of $T = 2000$ time-steps, number of areas involved in the fitness calculation $J = 9$, and number of random initial conditions per fitness evaluation $I = 30$. Figure 5 shows a typical convergence history of the GA.

At the end of 150 generations, we take the best CA lookup-table in the genetic algorithm population to be our solution. Figure 4 shows some snapshots of one solution, dubbed $\phi_{heap}$, in action. Starting from an initial random distribution of objects, the agents start by forming little piles which they eventually merge into one large heap. This strategy is similar, for example, to that employed by *Pheidole pallidula* (ants) in the clustering of corpses or *Leptothorax unifasciatus* (also ants) in the clustering of larvae (Deneubourg et al., 1990).

If one plots the fitness time series of the system starting from a random initial condition, a curve like the top plot in Figure 6 is the result. However, if one averages the fitness time series over say 1000 initial conditions, the much smoother curve in the bottom plot of Figure 6 results. Based on this average time series, an emergence time (Hanson and Crutchfield, 1995) may be calculated which represents how long the system takes, on average, to go to a steady state fitness (0.95 of maximum fitness). For the parameters used to evolve the $\phi_{heap}$ solution ($31 \times 30$ world, 30 agents, 60 objects), it should take about 5200 time-steps for the system to organize itself.
6 Scaling the Solution

Although we evolved the solution with one particular set of parameters (31 × 30 world, 30 agents, 60 objects) it is interesting to see what happens as these parameters change but the CA lookup-table remains the same. Figure 7 shows some snapshots of the system in action with a 91 × 90 world, 270 agents and 540 objects. Again, smaller piles of objects are formed first and eventually merged into a single large heap. Obviously it has taken much longer in this larger system for the single heap to form. This is not unreasonable especially if one takes the view that the collective is performing a computation; more complicated computations should take longer.

It was found that the most effective way to scale the solution was to keep the densities of agents and objects the same as those used to evolve the solution, namely 1/30 agents per lattice site and 1/15 objects per lattice site. Various problems were found to occur if these densities were changed drastically. They may change by reasonable amounts without much effect on system behaviour. If we keep the agent and object densities constant, the system may be adequately described by the size of the y dimension of the lattice grid (the x size is always equal to y size plus 1). Thus the y lattice size may be thought of as the characteristic problem size. In order to see the effects of scaling, this problem size was varied from 22 to 90 (recall the system was evolved on a problem size of 30). Figure 8 shows the maximum fitness obtained by the system in an extended 100,000 time-step run for each problem size. We can see that the system is able to achieve equally high fitnesses even for problem sizes as high as 90 (as in Figure 7). This is encouraging because for a problem size of 90, there are 540 objects which is almost an order of magnitude larger than the number for which the solution was originally evolved (60).

It is of interest to see how long each problem size took to achieve the final fitness. As before we can calculate an emergence time which is the amount of time to reach 0.95 of the fitness in Figure 8. Figure 9 shows how emergence time varies with problem size. It is interesting that this variation is somewhat linear.

7 Discussion

It is quite encouraging that our method of evolving global behaviours in a group of robots has been successful on the heap formation task. The approach is perhaps not immediately applicable to real world engineering projects but there is a growing belief that decentralized, self-organizing systems will have their place in future technologies. They might be used in environmental cleanups, surface explorations, or min-
Decentralized control could be very useful if so called nanotechnology is to ever become a reality. At the very least this experiment helps further the notion that these types of system must be designed with the whole system (agents plus environment) in mind. Each agent has no idea that it is helping to form heaps, it just follows the rules laid out for it. In fact, it would be very difficult to predict just what would happen to the objects by analyzing a single agent. As (Matarić, 1997) puts it:

"Since all control in such distributed systems is local, it scales well with the number of agents, does not require global communication, and is more robust to sensor and effector errors. However, global consequences of local interactions between agents are difficult to predict."

This is perhaps an understatement. However, it has been shown here that it is possible to evolve successful behaviours on a reduced problem size and then scale up the solution to the desired full problem size.

The work here was inspired by and follows on that of the EvCA group at the Santa Fe Institute. In their own words (Mitchell et al., 1996) their work is "a first step in employing GAs to engineer useful emergent computation in decentralized multi-processor systems. It is also a first step in understanding how an evolutionary process can produce complex systems with sophisticated collective computational abilities."

Although the work presented here is more plausible as an approach to collective robotics, the added complexity of the system makes its analysis quite difficult. We showed that, on average, our solution $\phi_{heap}$ produces the desired system behaviour after a certain emergence time. This is nice from an engineering perspective but does not further our general understanding of multiagent systems. Are there some basic principles of self-organizing systems? Even though the system presented here is barely complex enough to do anything useful, it is almost too complex to help us answer this question. We are certainly in need of a mathematical framework in which to couch the analysis of these types of systems; this may help us to give something back to our understanding of natural and biological systems whence our inspiration came. Computational mechanics (Hanson and Crutchfield, 1995; Feldman and Crutchfield, 1998), statistical mechanics and information theoretic approaches to multiagent systems...
are all candidates for this framework.

The last point to make concerns the difference between software agents and embodied or real world autonomous agents. (Pfeifer, 1996b) (among others) argues that the best way to study autonomous agents is in the real world since

...the available information is always “incomplete”. In fact, it is not even defined what “complete” information would mean. Expressed differently, the real world is only partially knowable. This implies that it is only partially predictable. Moreover, it is not defined what the possible moves are. In addition, the real world changes continuously, even if you don’t do anything.

It could be argued that many of these real world properties are also found in our simulated heap formation world but we tend to agree that just because an approach works in software it conveys little information about how well it will work in hardware. It should be noted that there have been successful heap formation studies in hardware (Pfeifer, 1996b; Maris and te Boekhorst, 1996). These studies typically use trial and error methods to develop local rules to produce the desired heap formation behaviour. They do show that heap formation is possible to achieve on small groups of real robots (5 to 10). Currently there are no plans to implement our method in a real world application; we believe that a great deal more can be learned through simulation at this point.

8 Conclusions

The approach to designing decentralized control for a robotic collective proposed here is somewhat different from many others found in the literature. We have shown that in the spirit of Santa Fe’s EvCA group, an evolutionary search may be used to discover local interactions between agents which produce desirable global behaviours. Rather than design the behaviour of each agent by hand and attempt to predict the global consequences, or worse yet accept whatever consequences arise, we have prespecified our desired global behaviour and allowed the agents to evolve their own local rules of behaviour. This paradigm of inverse design through optimization is not new but its application to multiagent systems is certainly not widespread. It is hoped that it may be applied to many other tasks and systems, both in software and hardware.

We hope that this contribution serves as a robotically plausible example of the design of a self-organizing system. The implications for biological systems are far from concrete at this point but it is interesting to see qualitatively similar clustering phenomena in insect societies. Perhaps some understanding of how natural systems are able to organize will result. Many researchers believe that these highly parallel and decentralized methods are the key to endowing artificial systems with intelligence. Our understanding of intelligence is certainly undergoing changes with everything we learn; we believe distributed systems will come to play an important role in that developmental process.

Acknowledgments

We would like to thank the Natural Sciences and Engineering Research Council of Canada and the Canadian Space Agency for supporting this work.
Bibliography


