# PATTERN FORMATION IN SWARM ROBOTIC SYSTEMS

Onur Soysal

Department of Computer Engineering
Middle East Technical University
06531 Ankara, Turkey
soysal@ceng.metu.edu.tr
http://www.kovan.ceng.metu.edu.tr

## 1 Introduction

This document summarizes my thesis studies on pattern formation in swarm robotic systems.

My current studies incline towards adaptation for obtaining pattern formation. Main problem considered is aggregation, getting a swarm of robots as close as possible. This problem is trivial for complicated robots with global communication. My environment on the other hand permits only local communication and limited computational capability. With these restriction the problem is much harder. Localization, map generation and high level agent communication is not possible. Robots are reactive with small degree of adaptative capability.

I have compiled literature for adaptation mechanisms for multiple robots. This survey is given in Section2. Adaptation is important since it might be the key point to build such limited robots with global behavior. Limitations in robot capability allows such robots to operate in many environments where other more complicated robots can't. Limited capabilities also allow cheaper robots, which is always desirable.

Designing controllers by hand is esspecially difficult with such tight constraints. So evolutionary robotics methodology is used. In this model controllers are optimized using evolutionary methods. This requires large amounts of computation. This problem is addressed by the platform described in Section3. This platform is presented in ISCIS'03 conferance and this section summarizes that work.

Last section discusses the preliminary results obtained from PES system and gives details of the plans on next stage of thesis.

## 2 Literature Survey

Robots in real world can't be equipped with complete world models because of limitations in capabilities of robots. Real world is also dynamic so robots should be able to modify models as needed. Adaptation can solve both these problems by automatically generating world models. These models can be modified as

environment changes since they are built using inference and experience rather than supervision.

Dudek et al. reports a taxonomy of collective robotics[18, 19]. This classification is important for understanding posibilities in multiple robotics. Their taxonomy consists of following axes:

- Collective Size
- Communication Range
- Communication Topology
- Communication Bandwidth
- Collective reconfigurability
- Processing Ability
- Collective Composition

One possible addition to this taxonomy could be adaptive capabilities, since adaptiveness differentiates the approach to the problem in many cases.

Cao et al. proposed another taxonomy[20] which includes adaptation as well. They differentiate studies according to following axes:

- Group Architecture
- Resource Conflict
- The Origin of Cooperation
- Learning
- Geometric Problems

This study surveys state of the art on multi-robot adaptation. The following section summarizes problems being solved using multi-robot learning approaches.

### 2.1 Problems

Multiple robots are not suitable for all problems. There are certain problems that require a single robot. In contrast, there are also tasks that require multiple robots, but these are relatively few in number. Between these two extremes there are tasks that can benefit from multiple robotics.

Current studies in multi robot learning concentrate on four such problems: Foraging, Robo-soccer, box pushing and pattern formation.

Pattern formation is the ability of robots to arrange themselves to desired relative positions. This problem is easy if global communication or centralized control is used, but for distributed decision making it is a considerable challenge.

In box pushing problem, robot teams are required to push a box to a goal position in minimal time. This operation requires coordination between robots. Increasing number of robots create increased amount of interference and lead to stagnation. This problem is first step for further complicated cooperation.

Foraging consists of exploring environment to find collectible items and bring them to a specified place. Foraging is a behavior common in insects which are biological proofs of feasibility of distributed decision maker systems. Foraging

may require cooperation among robots to carry items that can't be carried by a single robot. Cooperation can also be used to improve exploration efficiency.

Robo-soccer is perhaps the most complicated problem in collective robotics. Robot teams compete in a soccer game. This task requires high level strategies, collaboration and coordination. Environment of robo-soccer is highly dynamic, since opposing team is unpredictable. Adaptation can improve the performance of a team in this environment. There is domain knowledge in the field which should be included in the architecture, like tactics and game rules.

Adaptation can be achieved in two levels: group level and individual level. Group level adaptation models are inspired by genetics whereas individual adaptation models are inspired by psychology.

Behavioral psychology provides the concept of reinforcement. Reinforcement is defined as stimuli which increases or decreases tendency to act. Reinforcement learning paradigm uses this concept to generate policies. Next section details studies on this paradigm.

### 2.2 Individual Adaptation

**Reinforcement Models** It is usually impractical to have supervision in tasks that robots are required to perform. Learning models that work with positive and negative feedback signals are used to overcome this problem. Reinforcement learning and its variants are popular in this context since they only need a signal of appropriateness.

Reinforcement Learning models become useless when the state space is too large. Using multiple learning modules for different states instead of a single complicated learning module is one approach to solve this problem. Takayashi's work[9] is one such study. The problem studied in his work is a reduced version of robo-soccer challenge. Opponents are assumed to have different modes of operation each with a different policy. Modules consist of predictors and planners. Predictor predicts the next action of opponent based on its previous behavior. Planner on the other hand generates optimal move based on this prediction. Predictors compete for better accuracy and only best predicting module is reinforced. This creates specialized modules for different modes of operation of the opponent.

The sample problem used in this work is ball chasing in presence of a random moving opponent. The results show improvement over single module learning. Reinforcement learning converges to optimal policy given *sufficient* trials but it is often the case that these *sufficient* trials are too large to be feasible. Piao[10] proposes an improved reinforcement learning method to improve learning speed of learning. This method is combination of rule learning, reinforcement learning and *action level selection* which is basically behavior rules for specific states.

The rule base consists of instances that are states passed through a fixed interval. These instances are labeled after each epoch using information gathered through the epoch. These instances are then combined to create rules. These rules are used as a prohibitive guide to inhibit useless or harmful actions.

*Action level selection* is composed of hard coded rules to govern general strategy of robots. *Action level* is also fed into reinforcement level together with sensor data to generate the state information.

Finally reinforcement learning module uses sensory information and *action level* to generate state and learns to generate actions. Piao applies this method to the robo-soccer problem. He assumes only one agent is learning at a given time and reports improved performance on learning with multiple robots over standard Q learning.

Reinforcement learning is intended for single entities, so doesn't have any mechanisms to support cooperative behaviors. Tangamchit's work[16] investigates this problem. This work addresses distinction between action level and task level systems. To solve problems, action level systems generate reactive behaviors. On the other hand task level systems generate tasks composed of subtasks possibly distributed over multiple agents. Paper defines cooperation as a task level activity, where robots can share resources and duties.

Two different types of reward are considered in this work: global and local. In global reward scheme, reinforcement received by a unit is distributed to whole group. In contrast local reward scheme does not distribute reward among units. Two learning algorithms are considered: Q-Learning and Monte Carlo Learning. Q learning uses cumulative discounted rewards and Monte Carlo Learning uses averaging process to assess the value of each action in each state. Reward is same for each state action pair in the episode. This scheme is slower since it disregards the importance of latter actions in episode which are usually more effective in obtaining reward.

The case examined for this study is puck collecting behavior which a subclass of foraging problem. Robots are required to collect pucks and to deposit them into the bin. Each action has a negative reward except the action of depositing a puck. The field consists of a home region, which doesn't contain any pucks, a deposit bin, and pucks distributed around the region. Two heterogeneous robots are used for this task. The first robot moves and collects better in the region outside the home region. The second robot is limited in movement to home region but can accomplish bin deposit action more efficiently. Optimal strategy requires robots to cooperate and first to bring pucks into home region and second to deposit them. This requires task level learning.

Results indicate that task-level cooperation can't be learned well using local rewards or discounted cumulative rewards as in Q learning. In opposition global rewards coupled with average rewards result in cooperative policies for this task.

Reinforcement learning only requires feedback for applied sequence of actions to incorporate domain knowledge. This is usually incorporated by choice of reward functions. Mataric[14] discusses reward functions in a foraging task. Although single goals are mathematically simple to analyze, they cause problems through acquisition of behavior. Especially contingent and sequential behaviors are hard to convert into monolithic goal functions. Instead of this, separate goal functions are used, each describing a subgoal of agent. A second improvement is progress estimators. These estimators give a rough idea of how well a specific goal

is going on. These two improvements greatly increase the usage of domain knowledge in the topic (by designing appropriate subgoals and estimating progress of the subgoals). They also give much more reinforcement than standard methods, since not only the final goal but also intermediate steps are reinforced.

This improved method is tested on real robots working on a foraging task. Robots are to collect pucks and to deliver them to *home*. Robots are also responsible to be present at *home* at certain intervals. Robots are given some simple reactive behaviors to reduce state space of learning problem to a manageable size. These behaviors are collecting pucks when it is immediately before agent, avoiding obstacles and dropping pucks when at *home*. Experiments are compared with optimal policy generated by hand. Results indicate the benefit of both improvements purposed.

An interesting note in this paper is the interference caused by agents. Increasing number of learning agents has detrimental effect on general learning speed and convergence.

**Alternative Models** Parker's[22] L-ALLIANCE model uses multiple behavior sets and global communications to achieve cooperation. Each behavior set has a monitor. These monitors check required conditions for activation of behavior sets, also assess the capability of agent and other agents. Parker introduces two motivations: impatience and acquiescence. Impatience correspond to tendency to take a task being done by other robots and acquiescence describes tendency to give up a task to be performed by another robot. L-ALLIANCE architecture changes these motivational parameters during learning.

L-ALLIANCE architecture requires robots to broadcast current actions to other robots. This architecture assumes that when a robot declares an action, the changes in environment that can be caused by result of that action are attributed to that robot. This handles credit assignment problem.

L-ALLIANCE architecture is can handle heterogenous groups and can adapt to failures or changes in robot abilities which are desired properties. On the other hand, L-ALLIANCE requires global communication and makes a strong assumption to solve credit assignment problem.

Goldberg et al. [15] propose *Augmented Marcow Models* (AMM). AMM is a markov model improved by additional statistics about transitions. It is designed to learn the statistics of the environment rather than to generate a policy. AMM's assume action being performed can be known perfectly, so it is differentiated from Hidden Markov Models.

AMM's are first order Markov Models but they are built incrementally. This incremental building gives them ability to better approximate such higher order trasitions in the system. Their work combines AMM's with behavior based robotics [23]. Each behavior is monitored using AMM's with different time scales. This allows system to respond both slow and fast changes in the environment.

### 2.3 Group Adaptation

Reinforcement learning is by definition centralized which is inefficient to implement in multi-robot systems. Yanli's study[11] on opportunistically cooperative neural learning proposes a trade-off for centralized versus decentralized learning debate. In pure decentralized learning models each agent keeps its learning experience hidden from other agents. This seriously affects performance of the group since the experience can not be shared. Yanli solves this problem by adding 'opportunistic' search. This strategy is similar to survival of fittest concept in genetic algorithms. Less fit networks copy highly fit networks to improve their performance.

Yanli reports the comparison of three cases, central, distributed and opportunistically distributed. These cases are tested on searching task where agents are required to cover as much of a given space as possible avoiding multiple passes as much as possible. The best strategy clearly is one that utilizes cooperation. All agents act simultaneously and plan their movements ahead of action. Agents also share their plans with other agents. These plans are used to predict the next action of all other agents by each agent. Learning takes place in these predictors. When the next action of other agents can be predicted precisely reward can be calculated.

Results show that central learning is superior to all this methods in performance. However central learning has many problems in fault-tolerance and communication. OCL (opportunistically cooperative learning) performs almost as well as central learning and both perform remarkably better than distributed only case.

Nolfi and Floreano's[21] work show combination of learning and evolution can generate a better adaptation mechnasim then either alone. Learning improves performance of evolution by allowing adaptation to changes in life span of individuals. Learning also allows experience gained from environment to be better utilized. Perhaps most interesting effect of learning on evolution is guiding evolution as described by Baldwin(1892). Learning gives chance to suboptimal individuals to survive. But learning has a cost, so ultimately genetical knowledge replaces learned information.

Nolfi et al. reports two kinds of interaction between learning and evolution. In first kind both learning and evolution modify same data, like weights in neural network. This model although showing improvement in performance is not biologically plausible. In second kind of interaction genetic information controls only the rules and parameters of learning in the network. This is more similar to biological model and has generated interesting results. Some of the experiments have generated networks that continously change the weights. These changes don't reflect the change in behavior. This shows this system can find a *dynamic equilibrium*, rather than a *static equilibrium*.

Agah[12] combines both individual and group adaptation in his work. Agah uses so called *Tropism Architecture* to approach multi robot learning problem. Tropism architecture serves as a learning module between senses and actions. Each tropism is defined as a tendency to elicit a response for a given stimuli.

Tropism architecture keeps a list of learned tropisms (i.e. state, action, tendency pairs). Agents make decisions based on matching tropisms to current state. A stochastic process is used to determine which actions to apply biased on the tropism values.

Both kinds of learning are applied using this architecture. In individual learning scheme,the list of tropisms are updated based on feedback obtained from environment. These updates include adding a new valid action for current state, increasing tropism value for a pair which has been positively reinforced and changing action when an invalid or negatively reinforced action is encountered.

In population learning, the tropism lists for each agent is converted into variable length bit strings. Using these bit strings, a genetic algorithm is run. The fitness of each individual is calculated based on the rewards it received during individual learning. Results indicate success of this twofold method even in absence of reinforcement propagation as in Q-learning.

It is not always possible to have behaviors beforehand and even behaviors should be learned in certain cases. Hexapod locomotion is such a case. Parker[13] studies on learning a cooperative box pushing task in hexapod robots. The main problem he is facing is the locomotion problem, since moving hexapod robot requires more complicated operations than wheeled robots. For this task Parker purposed Cyclic Genetic Algorithms (CGA), which handles requirements of such complicated control. The motivation behind CGA's is evolving a sequence of operations instead of simple stimulus-response pairs. CGA encodes a series of activations which are to be repeated by the agent.

Fitness of each chromosome for a given task is calculated by using a computer simulation where the chromosome to be evaluated is paired with the best known solution to the problem. The success of the group is used as the fitness measure for the chromosome. Results indicate the effectiveness of purposed method.

Cooperation requires coordination among robots, which requires communication. Early approaches to cooperation used peer-to-peer communication models. This, although possibly required for optimal solution, requires increasing computational power and bandwidth for increasing number of robots in the system. Local communication reduces bottlenecks in communication but not totally solves this problem. Stigmergy, i.e., communication through environment changes, is a possible solution to communication bottleneck. This implicit communication scheme allows scalability and is observed in social insects.

Yamada's[17] work provides a working implementation of an implicit communication system for cooperation in robot groups. This scheme is applied to the box pushing problem. Goal is identified with a light source and robots are assumed to be capable of the following: detecting whether box being pushed is moving or not, presence of other robots and presence of walls. Here walls are modeled as unmovable boxes so they are ignored in the end. The authors generate situations to solve implicit communication problem. Situations abstract models of state of the world, which are computed using the sensor data and some very crude memory (such as counters for some sensor readings). Robots

have sets of rules for each situation. These rules are applied according to sensor readings.

## 3 Parallelized Evolution System

I have deveoloped Parallelized Evolution System together with Erkin Bahceci in Kovan Research Lab. This section describes this platform.

Evolutionary Robotics[6] is a new approach for creating autonomous robots. An active research track in Evolutionary Robotics is the development of behaviors for simulated robot systems. In these studies, the goal is to evolve controllers that uses the sensory information of the robot to control robot's actuators such that the robot accomplishes a desired task. Initially a population of genotypes that encode the controllers is given. Then the robot is simulated under the control of the controller specified by the genotype and its fitness is evaluated. The fittest controllers are then allowed to reproduce as define by a set of genetic operators, and the process is repeated. Evolving behaviors in simulated robot systems require a large amount of computational power, mainly due to the evaluation of different individuals. Therefore the computational requirements of using evolutionary methods tend to be proportional to the computational requirements of evaluating a single individual. This creates a major bottleneck for evolutionary methods.

Recently there have been projects[3, 5] that aim to create platforms that can use the idle processing power of computer systems that are connected over a network. SETI@home[3] is a scientific experiment that uses Internet-connected computers in the Search for Extraterrestrial Intelligence (SETI). The downloaded program runs as a background task or a screensaver getting chunks of radio telescope data from a server and performing a computation intensive processing of the data to seek signs of artificial signals and reporting the results back to the server. This platform made it possible for the SETI project to utilize a total computing power that equals or surpasses supercomputers.

This section reports the development of a platform, named *PES* (Parallellized Evolution System), that parallelizes an evolutionary method on a group of computers connected via a network. In the next subsection, we describe PES and its specifications. Section 3.1 describes the implementation of PES, explaining server and client components of the system. Section 3.2 describes the speedups obtained using PES for the problem of evolving behaviors for a swarm of simulated mobile robots. Section 4 summarizes the results and discusses the shortcomings and future development directions of the system.

### 3.1 PES

PES is a platform to parallelize evolutionary methods on a group of computers connected via a network. It separates the fitness evaluation of genotypes from other tasks (such as selection and reproduction) and distributes these evaluations onto a group of computers to be processed in parallel. PES consists of

two components; (1) a server component, named PES-Server, that executes the evolutionary method, the management of the communication with the client computers, and (2) a client component, named PES-Client, that executes programs to evaluate a single individual and return the fitness back to the server. Figure 1 shows the structure of a PES system.

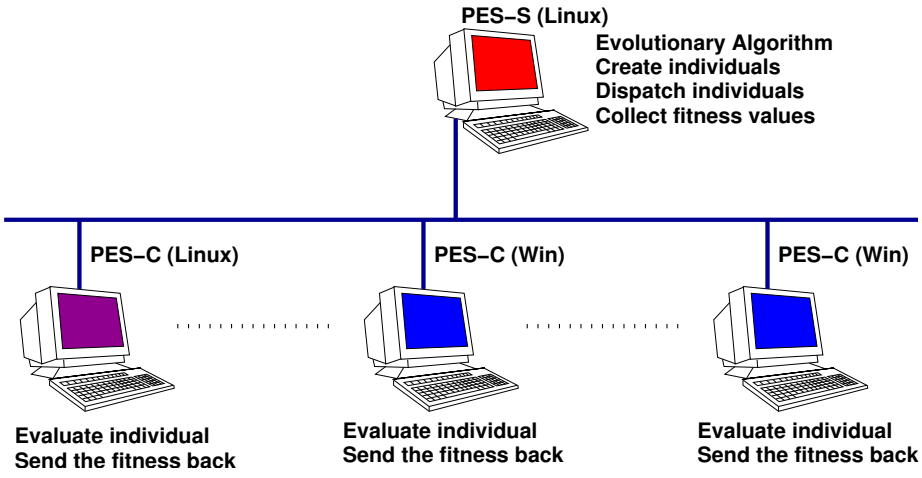



**Fig. 1.** Sketch of PES system is shown. The PES-Server runs on a Linux machine and handles the management of the evolutionary method. It executes the selection and reproduction of the individuals (genotypes) which are then dispatched to a group of PES-Clients (running both Windows and Linux systems). The individuals are then evaluated by the clients and their fitnesses are sent back to the server.

PES provides the user with an easy interface that relieves him from dealing with the communication between server and client processes. PES-Client is developed for both Windows and Linux, enabling the PES system to harvest computation power from computers running either of these operating systems. An easy-to-use framework for implementing evolutionary methods, and the interoperability of the system distinguishes PES from other systems available and makes it a valuable tool for evolutionary methods with large computational requirements.

PES uses PVM (Parallel Virtual Machine)[4][1], a widely utilized message passing library in distributed and parallel programming studies, for communication between the server and the clients. We have also considered MPI[2] as an alternative to PVM. MPI is a newer standart that is being developed by multiprocessor machine manufacturers and is more efficient. However PVM is more suitable for our purposes since (1) it is available in source code as free software

[1] Available at http://www.csm.ornl.gov/pvm/pvm_home.html.

and is ported on many computer systems ranging from laptops to CRAY supercomputers, (2) it is inter-operable, i.e. different architectures running PVM can be mixed in a single application, (3) it does not assume a static architecture of processors and is robust against failures of individual processors.

PES wraps and improves PVM functionality. It implements a time-out mechanism is implemented to detect processes that have crashed or have entered an infinite loop. It provides *ping*, *data* and *result* message facilities. Ping messages are used to check the state of client processes. Data messages are used to send task information to client processes and result packages are used to carry fitness information from clients.

Now we will describe the PES-Sever and PES-Clients.

**PES-Server** PES-Server provides a generic structure to implement evolutionary methods. This structure is based on Goldberg's basic Genetic Algorithm[1] and is designed to be easily modified and used by programmers. The structure assumes that fitness values are calculated externally. In its minimal form, it supports tournament selection, multi-point cross-over and multi-point mutation operators.

PES-Server maintains a list of potential clients (computers with PES-Client installed), as specified by their IP numbers. Using this list, the server executes an evolutionary method and dispatches the fitness evaluations of the individuals to the available clients. The assignment passes the location of the executable to be run on the client as well as the parameters that represent that particular individual and the initial conditions for the evaluation. Then it waits for the clients to complete the fitness evaluation and get the computed fitness values back.

PES-Server contains fault detection and recovery facilities. Using the *ping* facility the server can detect clients that have crashed and assign the uncompleted tasks to other clients. In its current implementation, the server waits for the evaluation of fitness evaluations from all the individuals in a generation before dispatching the individuals from the next generation.

**PES-Client** PES-Client acts as a wrapper to handle the communication of the clients with the server. It fetches and runs a given executable (to evaluate a given individual) with a given set of parameters. It returns the fitness evaluations, and other data back to the server.

Client processes contain a loop that accepts, executes and sends result of tasks. Client processes reply to *ping* signals sent by the PES-Server to check their status. Crashed processes are detected through this mechanism.

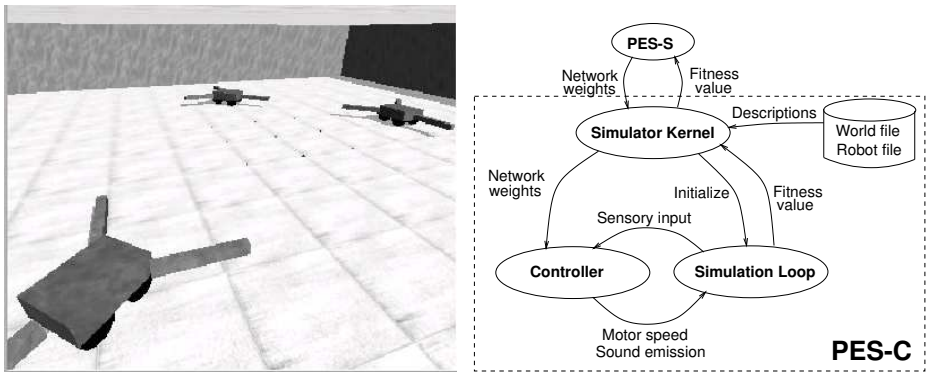
PES-Clients are developed for single processor PC platforms running Windows and Linux operating systems. Note that to use clients with both operating systems the fitness evaluating program should be compilable on both systems. In its current implementation, these clients have the fitness evaluation component embedded within them to simplify the communication. Yet, once the clients are

installed, the fitness evaluation component of the system can be updated using *scp* (secure copy) utility from the server.

### 3.2 Experimental Results

The PES platform is developed as part of our work within the Swarm-bots project[2][7], for evolving behaviors for a simulated swarm of mobile robots. We have conducted experiments to evolve behaviors for clustering a swarm of robots and analyzed the speed-up and efficiency of the PES system in this task.

The swarm of robots and their environment are modeled using ODE (Open Dynamics Engine), a physics-based simulation library, Figure 2(a). The parameters of this simulation and parameters of the controller (network weights) are passed to the PES-Client from the PES-Server. The simulator constructs the world and runs it, by solving their physical dynamic equations. Movements of the robots are determined by their controller as specified by the genotype. This controller uses the sensors of the robots and moves the robots for 2000 time steps. Then a fitness value is computed based on a measure of clustering achieved. This fitness value is then returned to PES-Server, Figure 2(b).




**(a)** **(b)**

**Fig. 2. (a)** A snapshot of the environment being simulated: Three mobile robots distributed in an arena are enclosed by walls can be seen. The rods emanating out of the robot bodies visualize the proximity sensing capabilities of the robots. **(b)** Architecture of PES-Client being used.

**The experimental set-up** We have installed PES-Clients to 12 PC's of a student laboratory at the Computer Engineering Department of Middle East Technical University, Ankara, Turkey. During the experiment, these computers

[2] More information can be found at http://www.swarm-bots.org.

are being used by other users and each of them had different workloads that varied in time. The population size is set as 48, requiring that 48 fitness evaluations take place during each generation. 30 generations were evolved.

Figure 3 plots the load of the 12 processors in time, during the evaluation of five generations. PES-Server waits for fitness evaluations of all the individuals in a generation before the selection and reproduction of the individuals of the next generation. In the plot, the vertical lines separate the generations. Within each generation, 48 fitness evaluations are calculated, which are visible as dark diamonds or dark horizontally stretched hexagons. It can be seen that the fitness evaluation time varies between different cases. There are two major causes of this. First, each processor has a different and varying workload depending on the other users of that computer. Second, the physics-based simulation of the swarm of robots slows down dramatically as robots collide with each other and the walls in the environment.

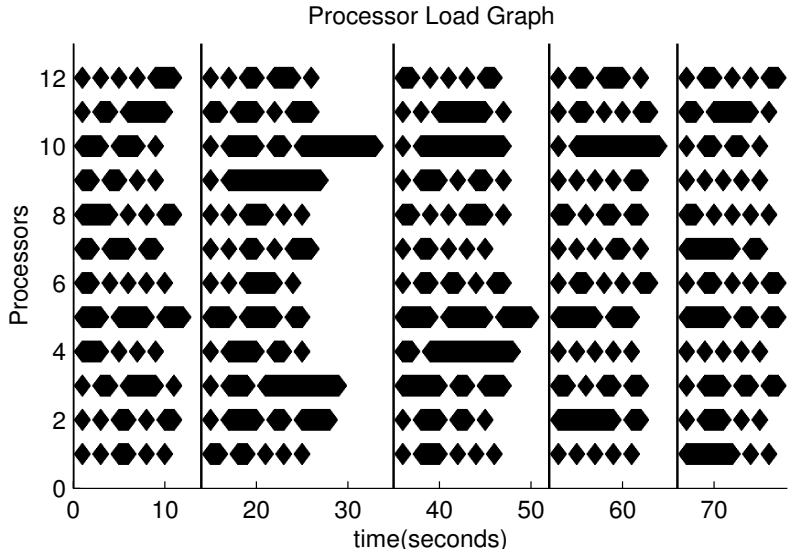


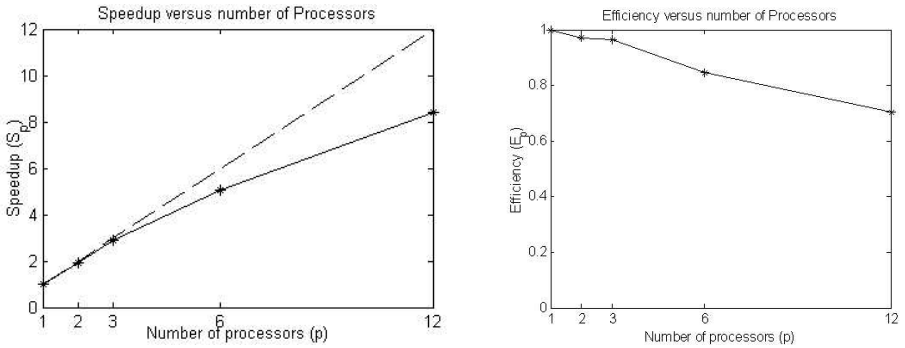

**Fig. 3.** The load of the 12 processors during 5 generations of evolution. See text for discussion.

In order to analyze the speed-ups achieved through the use of the PES system and its efficiency, we have repeated the evolution experiment using 1, 2, 3, 6 and 12 processors. The data obtained is used to compute the following two measures:

$$S_p = \frac{\text{Time required for single machine}}{\text{Time required for p machines}}$$

$$E_p = \frac{\text{Speed-up with p processors}}{\text{p}}$$

The results are plotted in Figure 4(a,b). Ideally $S_p$ should be linear to number of processors and $E_p$ should be 1. The deviance is a result of the requirement that all individuals need to be evaluated before moving on to the next generation. As a consequence of this, after the dispatching of the last individual in a generation, all-but-one of the processors has to wait for the last processor to finish its evaluation. This causes a decrease in the speed-up and efficieny plots. Note that, apart from the number of processors, these measures also depend on two other factors: (1) the ratio of total number of fitness evaluations in a generation to the number of processors, (2) the average duration of fitness evaluations and their variance.




**(a)** **(b)**

**Fig. 4. (a)** Speed-up is plotted against number of processors. **(b)** Efficiency is plotted against number of processors.

In Section 3.1, we had described a ping mechanism that was implemented to check whether a processor has crashed or not. This mechanism was crucial since we envision PES to harvest idle processing powers of existing computer systems and cannot make assumptions about the reliability of the clients. Figure 5 shows the ping mechanism at work during an evolution where we had 20 fitness evaluations in each generation run on 4 processors. Similar to the plot in Figure 3, this plot shows the loads of the processors. The numbers in the hexagons are labels that shows the number of the individual being evaluated. The continuous vertical bars separate the generations. the dotted vertical lines that are drawn at 15, 30, 45, and 60 seconds mark the pings that check the status of the processors. In this experiment, processor 2 crashed on while it is evaluating individual 5 after the first ping. PES-Server detected this at the second ping (at time 30), assigned the evaluation of individual 5 to processor 1, and removed processor 2 from its list.

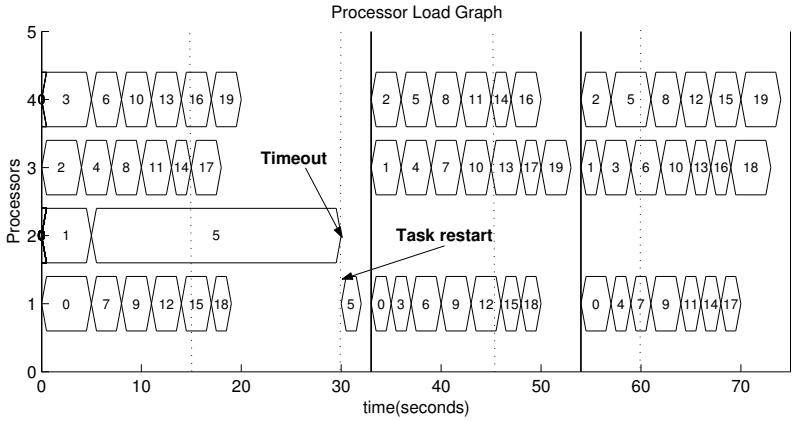



**Fig. 5.** Load of each processor during a run in which a processor fails. See text for discusssion.

## 4 Conclusion

In Section2 I have summarized current state of art in multi-robot adaptation. This study provides a starting point for my thesis which will focus on multi robot adaptation. Individual and group adaptation schemes are considered in this section, since swarm robotics may benefit from both.

Evolutionary robotics is a promising direction. Most notably works of Nolfi et al.[21] and Agah et al.[12] use evolutionary methods in adaptation.

In previous section I have reported a new parallel processing platform for running evolutionary methods. Unlike existing parallel processing platforms, PES provides a generic framework within which evolutionary methods can be easily implemented. PES provides an interface which relieves its user from the details of communication and the status of the clients. PES adds better fault tolerance mechanisms to PVM allowing crashed processes be detected. The user only needs to program a separate client program with a few constraints and should customize the server program to meet its requirements while leaving the rest to PES.

The analysis showed that although evolutionary methods lend themselves easily to parallelization, non-incremental selection and reproduction (that is the selection and reproduction occuring only after all the individuals are evaluated) is a major source of wasting processing time. The use of incremental selection and reproduction methods would eliminate this problem.

I plan to implement aggregation task with learning. This task can be solved partially with introduction of random inputs. My aim is to replicate and surpass these results with learning mechanisms. The problem of designing learning

controllers will be adressed using evolutionary methods. Evolutionary methods will run on parallel using PES library.

As of learning models I plan to use reinforcement models as baseline. These models are widely used in community and are well studied.

Neural networks models are common in evolutionary robotic literature and they will be included in my study. They are suitable for both evolution and learning. So they will be very convinient for my task.

Threshold models on the other hand are common in swarm robotics practice. These models are simple yet can yield complex results. These models have the benefit of biological plausibility. There's not much work on adaptive threshold models. One of the aims of this thesis will be inclusion of adaptive threshold models.

These three main catagories of models will be compared in performance. Also a parallel research by Erkin Bahceci, which studies on effect of probablistic inputs in pattern formation will be considered in this comparison.

## Acknowledgments

This work was partially funded by the SWARM-BOTS project, a Future and Emerging Technologies project (IST-FET) of the European Community, under grant IST-2000-31010. The information provided is the sole responsibility of the authors and does not reflect the Community's opinion. The Community is not responsible for any use that might be made of data appearing in this publication.

## References

1. Goldberg D. E., *Genetic Algorithms in Search, Optimization, and Machine Learning.* Reading, MA: Addison-Wesley, 1989.
2. Hempel, R., *The MPI Standard for Message Passing.* High–Performance Computing and Networking, International Conference and Exhibition, Proceedings, Volume II: Networking and Tools. Ed. Gentzsch, Wolfgang and Harms, Uwe. 247–252,1994.SV, LNCS vol797, 1994.
3. Korpela E., Werthimer D., Anderson D., Cobb J., Lebofsky M., *SETI@home: An Experiment in Public-Resource Computing* Communications of the ACM, Vol. 45 No. 11, pp. 56-61, November, 2002.
4. Kowalik J. (ed), *PVM: Parallel Virtual Machine: A Users' Guide and Tutorial for Networked Parallel Computing.* MIT Press Scientific and Engineering Computation, 1994.
5. Mukherjee S., Mustafi J., Chaudhuri A., *Grid Computing: The Future of Distributed Computing for High Performance Scientific and Business Applications* Lecture Notes in Computer Science, Vol. 2571, pp 339-342, 2002.
6. Nolfi S., Floreano D., *Evolutionary Robotics: The Biology, Intelligence, and Technology of Self-Organizing Machines*, MIT Press/Bradford Books, 2000.
7. Şahin E., Labella T.H., Trianni V., Deneubourg J.-L., Rasse P., Floreano D., Gambardella L.M., Mondada F., Nolfi S., Dorigo M., *SWARM-BOT: Pattern Formation in a Swarm of Self-Assembling Mobile Robots.* In A. El Kamel, K. Mellouli, and P.

Borne, editors, Proceedings of the IEEE International Conference on Systems, Man and Cybernetics, Hammamet, Tunisia, October 6-9, 2002. Piscataway, NJ: IEEE Press.
8. Vinschen C., Faylor C., Delorie D. J., Humblet P., Noer G., *Cygwin User's Guide.*
9. Takahashi, Y. Edazawa, K. Asada, M. *Multi-module learning system for behavior acquisition in multi-agent environment*, Intelligent Robots and System, 2002. IEEE/RSJ International Conference on ,927- 931 vol.1.
10. Piao S, Hong B., *Fast reinforcement learning approach to cooperative behavior acquisition in multi-agent system.* Intelligent Robots and System, 2002. IEEE/RSJ International Conference on,871- 875 vol.1.
11. Yanli Y., Polycarpou M.M., Minai A.A., *Opportunistically cooperative neural learning in mobile agents*, Neural Networks, 2002. IJCNN '02. Proceedings of the 2002 International Joint Conference on ,2638-2643.
12. Agah A., *Phylogenetic and Ontogenetic Learning in a Colony of Interacting Robots*, Autonomous Robots 4(1):85 - 100.
13. Parker, G.B. Blumenthal, H.J., *Punctuated anytime learning for evolving a team*, World Automation Congress, 2002. Proceedings of the 5th Biannual,559- 566.
14. Mataric' M.J., *Reward Functions for Accelerated Learning*,Machine Learning: Proceedings of the Eleventh International Conference, 1994, 181-189.
15. Goldberg D., Mataric M.J., *Maximizing Reward in a Non-Stationary Mobile Robot Environment*, invited submission to the Best of Agents-2000, special issue of Autonomous Agents and Multi-Agent Systems, 6(3), 2003, pp. 281-316.
16. Tangamchit P., Dolan J.M., Kosla P.K., *The necessity of average rewards in cooperative multirobot learning*,Robotics and Automation, 2002. Proceedings. ICRA '02. IEEE International Conference on,1296- 1301 vol.2.
17. Yamada, S., Saito J., *Adaptive action selection without explicit communication for multirobot box-pushing*,Systems, Man and Cybernetics, Part C, IEEE Transactions on,Volume: 31, Issue: 3, 398-404.
18. Dudek, G., Jenkin, M., Milios, E., Wilkes, D., emphA Taxonomy for swarm robotics, Proceedings IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS), Yokohama, Japan, July 1993, pp. 441-447.
19. Dudek G., Jenkin M., Milios E., *A Taxonomy of Multirobot Systems*, Robot Teams: From Diversity to Polymorphism, Balch T., Parker L. E. (Eds), 2002, A.K. Peters.
20. Cao Y. U., Fukunaga S. A., Kahng B. A., *Cooperative Mobile Robotics: Antecedents and Directions*, IEEE/TSJ International Conference on Intelligent Robots and Systems, Yokohama, Japan, 1995.
21. Nolfi S., Floreano D., *Learning and Evolution*, Autonomous Robots, 7(1): 89-113, 1999.
22. Parker L. E. *Lifelong Adaptation in Heterogeneous Multi-Robot Teams: Response to Continual Variation in Individual Robot Performance*, Autonomous Robots, 8(3):239 - 267 , 2000.
23. Brooks R. A. *Intelligence without representation*, Artificial Intelligence, 47, 139-159, 1991.