Genetic Algorithms for Automated Negotiations: A FSM–Based Application Approach

M.T. Tu, E. Wolff, W. Lamersdorf *
Distributed Systems Group
Computer Science Department
University of Hamburg
Vogt-Kölln-Str. 30, 22527 Hamburg, Germany
[tu|3wolff|lamersd]@informatik.uni-hamburg.de

Abstract

In this paper, an approach to implement strategies for automated negotiations in electronic commerce applications is presented. It is based on genetic algorithms (GAs) that evolve FSMs (Finite State Machines). Each of these FSMs represents a negotiation strategy that competes against other strategies and is modified over time according to the outcome of this competition by using GA principles. The paper gives an overview of negotiating agents and work related to this paper. Then the application of Genetic Algorithms to FSMs is presented and relevant details on the implementation are given.

1 Introduction

Software agents are widely recognized to be the basis for building the information infrastructure of the emerging 'Net' society. This is at least true for the deployment of mobile agents in electronic commerce. Being the central interaction scheme in economic activities, negotiation is one of the main focuses in the development of software agents to perform online commercial transactions. To enable the development of agents that can negotiate with each other or with humans, a negotiation framework for mobile agents was proposed in [5, 6]. But concrete negotiation strategies need to be developed that can be integrated into this infrastructure.

A good strategy should be able to find the optimal outcome of the negotiation for all partners. This is often not achieved by human negotiators and thus 'money is often left on the table' due to an suboptimal negotiation outcome, especially in complex multi-issue negotiations [4]. Negotiation can be considered a search for an optimal negotiation outcome with respect to the utility functions of each partner. This function returns a value representing the gain from a specific negotiation outcome for the partner. Each negotiation partner has only knowledge of his utility function, because to let the partner know the utility function would often be a disadvantage. Another problem is that in many cases, the search space of all possible negotiation outcomes has a too high dimension to use analytic optimization to find the optimal solution. Problems with a large search space with little information on its structure available are known to fit well to the use of genetic algorithms. Thus, it was realized that the implementation of strategies using genetic algorithms (GA) might be a very fruitful approach to support automated negotiations. In this paper, we present the implementation of a GA framework which is based on Finite State Machines (FSM) and show how it can be applied to bilateral negotiations. The remainder of the paper is organized as follows: In Section 2, the basic idea of applying GAs to negotiations is described including hints to work presented by other authors. Section 3 describes how to design GAs which use FSMs as the basic data structures. Section 4 presents the current implementation of the framework, its application to bilateral negotiations and as well as relevant results of experiments done. Finally, some open issues are discussed in Section 5.

2 Genetic Algorithms for Automated Negotiations

GAs are inspired by the evolution taking place in nature. Evolution in nature discloses an unmatched variety of different species each optimized for its ecological niche. This is achieved by very simple principles: selection together with reproduction, crossover and mutation. Selection

* This work is supported, in part, by grant no. La1061/1-2 from the German Research Council (Deutsche Forschungsgemeinschaft, DFG).

means that only the fittest individuals survive. Reproduction is the ability to breed new individuals and mutations are deviations during this reproduction process. Crossover is the ability to take two individuals (parents) to breed one new individual that shares some attributes with each parent.

In GAs, the basic principles of evolution (reproduction, mutation and crossover) are used to create objects that are optimized for a certain function. To carry out this process, a set of objects (the *population*) is evaluated at discrete points in time (between the *generations*). Each individual has a certain probability to be taken into the next generation. This probability depends on its quality (*fitness*) measured by the function that should be optimized. The individual can be propagated into the next generation either unchanged (reproduction), mutated or as resulted from a crossover with another individual. The evolutionary approach can be used for the optimization of numerical problems (*Genetic Algorithms*) as well as for the automatic generation of programs (*Genetic Programming*). The main benefit of evolutionary techniques is that they make few assumptions about the environment they are applied to. All that is needed is the definition of the reproduction, mutation and crossover operations for the data structures used and a fitness function. Thus, evolutionary techniques are very often used successfully in environments where little knowledge of their structure exists.

In the field of electronic commerce, the need for good negotiation strategies is very obvious. However, there are only a few concrete trials to apply GAs for negotiation, most notably by Oliver [2]. In Oliver's experiments, strategies are modeled as simple sequential rules made up of offers separated by thresholds which represent the total utility value of an offer. Offers themselves are modeled as tuples of values corresponding to negotiable attributes (e.g., price, quality, delivery etc.), each of which has a certain utility value (see Figure 1).

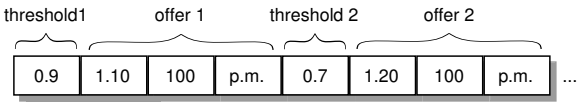

Figure 1. Example of an evolution-based strategy (following [2]).

System operation begins with building a population of random strategies. Then successively, a strategy at a time is taken to take part in a negotiation and its payoff is calculated when the negotiation ends. After such a population has been tested in this manner, a new one is produced by selecting the strategies with the best pay-offs and applying genetic operators such as mutation and crossover on them to generate new ones filling the new population. When this process is iterated a number of times using different strategies to perform against each other, a certain learning effect is achieved.

The problem with this model is that it does not even offer enough expressiveness to model strategies with simple decision rules such as 'If my negotiation partner has only made offers with a small payoff for me so far, I won't make such offers myself'. To express such strategies, a model with branching and some kind of memory is needed. Successful strategies even in so simple game theoretical experiments as the iterated prisoners' dilemma [3] usually make use of such elements.

3 Using GAs on FSMs

Due to the shortcomings in Oliver's representation of strategies, a different representation format was needed. So it was decided to model negotiation strategies as Finite State Machines (FSMs) because they offer a notion to model the branching and memory that are lacking in Oliver's approach. Branching can be done by edges leading to different states and memory can be achieved by using states. Yet, the model chosen is not as complicated as the trees or linear genomes usually used by Genetic Programming and thus the search space is considerably smaller.

Usually, FSMs just accept regular sets. But in order to be applied to negotiations, they do not need to accept a set but instead must generate some output (offers and other negotiation actions) as response to some input (offers from the negotiation partner). This can be achieved by using FSMs with output such as Mealy automata.

The Mealy automata used for negotiation strategies have specific attributes:

One initial state Thus, the start of the strategy is set deterministically.

No final state The end of a negotiation should rather be modeled by a special output or should be ended by the negotiation control after a certain number of negotiation steps.

Complete For each state and each input symbol, there must be at least one edge. Otherwise, it would be possible that during a negotiation, a FSM stops because it gets an input symbol that it can not react to.

Deterministic For each state and each input symbol, there must be at most one edge.

By using one initial state and making the automaton deterministic, the payoff of two strategies negotiating with each other can be calculated deterministically, i.e. every negotiation between the same two strategies will always have the same outcome.

Now, the three genetic operators reproduction, mutation and crossover need to be defined. Reproduction is trivial: the FSM just needs to be copied. However, the other two operators are more demanding and are presented in the next section.

3.1 Mutation and Crossover of FSMs

To mutate an FSM, several different operators can be chosen from. These include changing the target or source of an edge, changing the output or input symbol of an edge and adding or deleting a state or an edge. In the implementation of the presented work, all the operators defined here are implemented and the decision which operator to use is made randomly. To maintain the characteristic of the FSM (complete and deterministic) when applying one of the operators, additional work must be done by the operators that is not apparent. So for example, if the source of an edge or the input of an edge is changed, there are some inputs that do not have an edge assigned to them any more. Thus, other edges need to be assigned to these inputs.

The main reason to use crossover as a genetic operator is to spread good parts of the genome of an individual in the population: It is the only operator that actually interchanges parts of the genome of two individuals by creating offspring from two parents that have parts of the genome of both. For FSMs, the crossover mechanism was defined as follows:

1. Divide the set of states of both parents in two subsets.
2. States that have an edge leading to a state in the other subset are called 'outputs'. States that have an edge leading to them from a state in the other subset are called 'inputs'.
3. Create the set of states of the children by joining the subset of states including the initial state of one parent with the subset not including the initial state of the other parent.
4. Connect the inputs and outputs of the two subsets.

The idea behind this algorithm is to separate a part of the FSM from the rest and put it into a different FSM. This is inspired by the graph crossover as described in [1], p. 124f.

3.2 Fitness

The fitness function gives a value that indicates how good an individual satisfies the optimization function. For the negotiating agents, the fitness value is calculated as follows: Each agent must have a utility function that provides his own benefit for each possible negotiation outcome. To calculate the fitness of the population, each agent undergoes a certain number of negotiations and the fitness value is calculated as the average of the utility values of the reached negotiation outcomes. It is important to choose the right number of negotiations for the calculation of the fitness value. On the one hand, it should be not to low to give an indication of how the strategy behaves when facing different other strategies. On the other hand, it should not be to high so that no computing power is wasted.

4 Current Implementation

For the prototype implementation, an object-oriented design approach was chosen and the Java programming language was used. However, the goal is to study the basic techniques described here. These can be considered independent of any concrete implementation.

4.1 GA Framework

The first step of the implementation was to create a generic framework for Genetic Algorithms.

An interface `Evolvable` was defined that must be implemented by those objects that should be used with the genetic algorithm. It defines the three genetic operators that can be applied to such objects (Mutation, Crossover and Reproduction). To offer the developer the possibility to implement these three operators in one class each, the class `AggregateEvolvable` was created. Therefore the developer can choose to implement the data structure including all genetic operators in one class or implement each one in its own class.

The `GPEvolver` class implements a generic Genetic Algorithm. The algorithm uses a (μ, λ) evolution strategy, i.e. the generation has a certain size μ, and then λ individuals are created from them ($\lambda > \mu$) by using the genetic operators. The parent individuals are chosen with a probability proportional to their fitness and the operators are chosen randomly. From the new population of size λ, the μ individuals with the highest fitness are propagated into the next generation as parents.

To calculate the fitness values, a class implementing the `Fitness` interface is used. This interface defines a method that takes two individuals as input and gives two fitness values for the two individuals as output.

This description gives only a short overview of the complete framework. However, note that this part of the framework is completely independent of the data structure that is used. Besides FSMs, it can be used for numerical optimization problems and in fact this was done as a test for the framework. This enables a separation of concerns: the generic framework can be extended with new algorithms and optimizations while the data structures stay the same.

4.2 Integration of FSMs

The main goal of the implementation of the FSMs is to make them as generic as possible. Thus, a FSM itself only accepts integers as inputs and also only returns integers as output. In this way, it is possible to generate FSMs that use an arbitrary finite set as input and output alphabet without any change in the FSM class itself. The gained genericity allows for the application of the conception of GAs working on FSMs for different purposes such as the optimization of `Protocol Adapters` (see [7]) very easily and also allows an easy adaptation of the FSM to different negotiation scenarios.

To implement the crossover operation, a class `FSM-Crossover` was implemented while the mutation as described was implemented by the class `FSMMutator`. The operators were described in more detail in section 3.1.

The `FSMFactory` class creates new FSMs with a given size of the input and output alphabets and a given maximum number of states. Because the `FSMFactory` makes use of the `AggregateEvolvable` design, each new individual must have a reference to instances of `Crossover` and `Mutator`. Thus, the `FSMFactory` also needs to have these references to create new `AggregateEvolvables` including an FSM as data structure.

4.3 Modeling Bilateral Negotiations using FSMs

To use the FSM model proposed above for negotiation strategies, the input and output alphabet need to be mapped to offers in the negotiation. Besides, testing scenarios need to be defined.

For the negotiation strategies, the output alphabet was chosen to be an encoding of the set of possible offers plus one additional element to accept the last offer and thus end the negotiation. The input alphabet was chosen to be an encoding of thresholds. This closely resembles the approach taken by Oliver (see [2]) but instead of just one threshold, there are multiple thresholds leading to different new offers. So a counter offer can be a specific response to the last offer.

One example for this encoding of negotiations might be as follows: the negotiation is about two issues s_1 and s_2, each can have a value from 0 to 5. Then, the output alphabet has 37 elements with 0–35 representing a counter offer (s_1, s_2) with s_1 = value / 6 and s_2 = value mod 6. 36 represts the acceptance of the last offer.

Assume that each agent has a utility function f that assigns a value between 0 and 1 to each offer o. Then, a possible definition of the input alphabet is : **0** $f(o) < 0.25$, **1** $f(o) \in [0.25, 0.5[$, **2** $f(o) \in [0.5, 0.75[$ and **3** $f(o) \in [0.75, 1.0]$.

So if a FSM reaches the state shown in the middle of Figure 2 and gets an offer with $f(o) < 0.25$, it returns a counter offer tuple of (2,4); if it gets an offer with $f(o) \in [0.75, 1.0]$, it accepts and so on. Additional optimizations must eliminate exceptional situations such as that an agent makes a counter offer with a lower utility value instead of accepting the original offer.

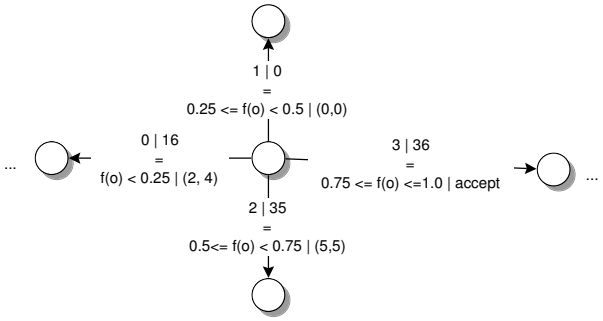

Figure 2. A part of a concrete FSM that implements a negotiation strategy. Each edge is labeled with the input and output symbol separated by | and their respective semantic.

To calculate the fitness, a function is implemented that takes two individuals and initiates a negotiation between them. For each individual, this function is called with several different other individuals by the GA framework. The resulting payoffs are used to calculate the fitness of each individual. This is actually the only part of the system where the integers the FSM operates on are mapped to their semantics.

4.4 Experiments and Results

Using this approach, we have performed comparative experiments by implementing the bilateral negotiation scenarios in the work of Oliver [2] using our FSM-based GA framework instead of Oliver's linear genomes. This work presents the following scenarios:

No Conflict Both agents have identical utility functions, i.e. for both the optimal outcome of the negotiation is identical.

Pure Distributive Every advantage for one agent is a disadvantage for the other.

Simple Integrative Here, the utility functions are contrary but the negotiation issues are of different relevance to each negotiation party. So a compromise needs to be found that fulfills the preferences of the negotiation parties as well as possible.

Divorce This resembles the scenario above but is more complex w.r.t. the number of negotiation outcomes.

For these scenarios, the fitness is defined by a utility function. So a certain fitness value for a population means that the agents in the population achieved that value as the mean value for the utility of the negotiation outcomes.

For the Genetic Algorithm, we used the parameters chosen by Oliver but to get mean values, 100 instead of the 10 runs in Oliver's work were used. Also the crossover and mutation probabilities were adjusted to get the best results possible. Even though Oliver gives values for both the mutation and crossover probabilities, it is not mentioned in his work why these specific values were chosen. Table 4.4 gives an overview of the results of the FSM-based experiments compared to those based on the genomes used in Oliver's work.

Scenario	FSM Agent1	FSM Agent2	Oliver Agent1	Oliver Agent2
No Conflict	0.86	0.86	0.88	0.88
P. Distributive	0.51	0.49	0.54	0.46
S. Integrative	0.59	0.57	0.57	0.57
Divorce	0.58	0.55	0.57	0.50

Table 1. Fitness comparison between this work and the work by Oliver.

These results mean that the following conclusions from Oliver's work can be transferred to FSMs: FSMs do better than random strategies. and FSMs can learn strategies using Genetic Algorithms. Also, it can be concluded that FSMs do as good as Oliver's linear genomes. However, to answer the question whether FSMs have advantages over Olivers linear genomes, the size of the input alphabet was modified. The fitness values given above were achieved using an input alphabet of size 4. Nevertheless, the values stay the same when the size of the input alphabet is modified; even if the input alphabet has a size of 1. But this means that in every state, there is one edge leading away from that state and thus the FSM can not make any decisions depending on the incoming offer. Thus, regarding Oliver's scenarios, the decision capabilities of FSMs do not pay off. However, these results are not very surprising as in Oliver's scenarios, the agents are only confronted with one kind of utility function during the whole evolution. So a data structure such as a FSM that makes it possible to react to *different* utility functions is not an advantage here. But if the agents in Oliver's scenario are optimized for a specific kind of utility function, they will reach only suboptimal results in negotiations with other kinds of utility functions. This also means that the agents do not learn generic strategies but rather strategies for a certain kind of utility function. So in more realistic scenarios, FSMs might very well offer better results than Oliver's approach.

5 Summary and Outlook

In this paper, a new approach to enable agents in an electronic marketplace to learn negotiation strategies was presented. First, a general introduction to Genetic Algorithms and the negotiation framework was given. Then, previous approaches to the problem using Genetic Algorithms were presented and the shortcomings were described. A new approach using FSMs as the basic data structure was proposed, the prototype implementation was described and first results from experiments with the implementation were given and discussed.

Although the results achieved with the implementation are quite promising, we still see chances for improvement: The results presented here clearly show that FSMs are capable of reaching good negotiation results. However, one of our goals is to show that FSMs are really superior to the data structures used so far. To decide this, more complex scenarios need to be implemented that can make use of the decision capabilities of FSMs. Such scenarios should also focus more on the applications in real world systems.

Another major goal is to integrate the Genetic Algorithm approach as a basic development pattern into a common mobile agent framework such as that described in [5] to make it available to a larger community and thus gain more experience with Genetic Algorithms. The GA approach could then also be compared to different approaches to make agents learn negotiation strategies.

References

[1] W. Banzhaf, P. Nordin, R. E. Keller, and F. D. Francone. *Genetic Programming — an Introduction: On the Automatic Evolution of Computer Programs and its Applications*. Morgan Kaufmann Publishers, 1998.

[2] J. R. Oliver. *On Artificial Agents for Negotiation in Electronic Commerce*. PhD thesis, The Wharton School, University of Pennsylvania, 1996.

[3] W. Poundstone. *Prisoner's Dilemma — John von Neumann, Game Theory, and the Puzzle of the Bomb*. Oxford University Press, 1993.

[4] H. Raiffa. *The art and science of negotiation*. Harvard University Press, 1982.

[5] M. T. Tu, F. Griffel, M. Merz, and W. Lamersdorf. A plug-in architecture providing dynamic negotiation capabilities for mobile agents. In K. Rothermel and F. Hohl, editors, *Proc. 2. Intl. Workshop on Mobile Agents, MA'98, Stuttgart*. Springer LNCS, 1998.

[6] M. T. Tu, C. Seebode, F. Griffel, and W. Lamersdorf. A Dynamic Negotiation Framework for Mobile Agents. In *Proc. 3. Intl. Symposium on Mobile Agents, MA'99, Palm Springs, California*. IEEE, 1999.

[7] D. M. Yellin and R. E. Strom. Collaboration specifications and component adapters. Technical report, IBM T.J. Watson Research Center, 1995.