CLUSTERING AGENT OPTIMIZATION RESULTS IN DYNAMIC SCENARIOS

André Restivo [a] Luís Paulo Reis [ab]

[a] *Faculdade de Engenharia da Universidade do Porto*
Rua Dr. Roberto Frias 4200-465 Porto Portugal
[b] *Laboratory of Artificial Intelligence and Computer Science*
Rua do Campo Alegre, 823 4150-180 Porto Portugal
{*arestivo,lpreis*}*@fe.up.pt*

Abstract

The application of optimization algorithms to parameter driven simulations and agents has been thoroughly explored in literature. However, classical optimization algorithms do not take into account the fact that simulations normally have dynamic scenarios.

This paper analyzes the possibility of using the classical optimization methods, combined with clustering techniques, in order to optimize parameter driven agents, in simulations having dynamic scenarios.

This will be accomplished by optimizing the agents in several random static scenarios and clustering the optimum results of each of these optimizations in order to find a set of typical solutions for the agent parametrization problem. These typical solutions can then be used in dynamic scenario simulations as references that will help the agents adapt to scenario changes.

The results of this approach show that, in some cases, it is possible to improve the outcome of simulations in dynamic environments while still using the classical methods developed for static scenarios.

1 Introduction

Agents normally have parameters that can be changed in order to tweak their performance. Finding the parameters that yield the best results is a recurrent problem that anyone developing agents has already faced.

Several optimization algorithms have been developed and studied over the years such as: Simulated Annealing, Tabu Search and Genetic Algorithms [7]. However, most of the times, these algorithms will only optimize agent parameters for a certain static scenario [1]. If the scenario being analyzed by the agents changes, the optimum parameters will probably also change.

Simulation scenarios are often also defined by a set of parameters. In cases where this does not happen, parameters describing the scenario can sometimes be extracted. In this way, it can be said that a simulation normally has two sets of parameters that will influence its outcome: environment, or scenario, parameters (that are normally out of the agent's control) and agent parameters that can be changed in order to get better results.

One way of optimizing agents *living* in dynamic simulations, would be to run the optimization process against several different static scenarios and then have the agent constantly use the optimum parameters found for the scenario that most resembles the current one. This solution has several drawbacks like, for example, the overhead caused by the constant changing of parameters. The proposed way, described in this paper, of tackling this problem is to use clustering algorithms in order to minimize the number of different parameter sets, thus minimizing the number of times parameters need to be changed without losing responsiveness or efficiency. We will show that, in certain simulations, using this minimized set of parameter configurations can improve agent performance.

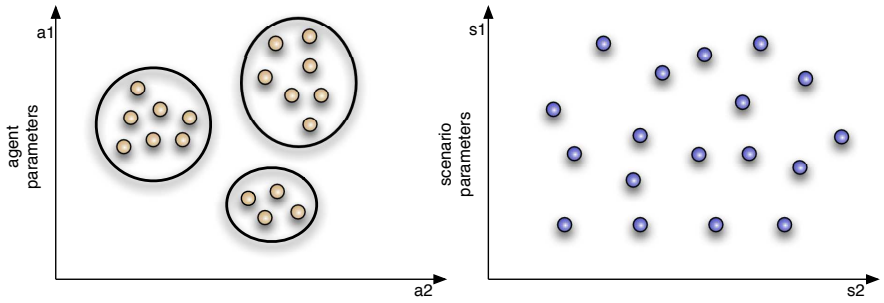

Figure 1: Optimum Agent Parameters per Scenario (clustered)

2 Clustering

Clustering can be defined as grouping sets of elements that are similar in some way. A simple way of achieving this objective is by obtaining maximum intra-cluster similarity and maximum inter-cluster dissimilarity.

Clustering methods have to deal with two different problems: membership (whether an element belongs to a certain cluster or not) and how many clusters to create. Most methods only deal with the first of these problems but some strategies have been developed to determine the number of clusters and cluster membership at the same time [3].

To develop an agent optimization system that can cope with scenario changes, solutions from an optimization problem must be grouped together, having in mind that different solutions apply to different initial conditions of the simulation. The objective is to find out which different kinds of solutions exist and to which type of problems each one of them can be applied (see Figure 1). To implement the aggregation of scenarios, the K-Means clustering algorithm was used [4].

The K-Means algorithm starts with the creation of k clusters with initial centroids estimated using some kind of heuristic. Elements are then assigned to the cluster with the nearest centroid (usually using simple euclidean distance measurements). New cluster centroids are then calculated and the process repeats until convergence is met (i.e. no element changes cluster).

The major drawbacks of this method are: the fact that the number of clusters must be predetermined; having a poor performance as distances to the cluster centroids must always be recalculated in each step; and its results being very dependent of the initial choice of cluster centroids.

3 Scenario Clustering

Optimizing n different scenarios for a simulation S, will produce a set of results $\overrightarrow{R}$, in the form $\overrightarrow{R} = (r_{1,\ldots,}r_n)$. Each one of those results r is the union of three other values $r = (\overrightarrow{s}, \overrightarrow{p}, v)$, where $\overrightarrow{s}$ is a set of environmental parameters, in the form $\overrightarrow{s} = (s_1, \ldots, s_m)$, $\overrightarrow{p}$ is a set containing the agent parameters that optimize that scenario, in the form $\overrightarrow{p} = (p_1, \ldots, p_k)$, and each v is the best value achieved in that optimization. The values n, m, and k represent, respectively, the number of different scenarios optimized, the number of different environmental parameters and the number of different parameters optimized in each scenario.

If we take all $\overrightarrow{p}$ from the set $\overrightarrow{R}$ we get a set $\overrightarrow{P}$ containing all good solutions for the problem (although each solution is optimum only for a specific scenario). Having the same approach we can derive a set $\overrightarrow{S}$ from all the $\overrightarrow{s}$ values from $\overrightarrow{R}$.

> **Hypothesis:** In certain simulations, given a set $\overrightarrow{P}$, containing the optimum solutions for a set of representative scenarios $\overrightarrow{S}$, it is possible to construct subsets $\overrightarrow{P}_1, \ldots, \overrightarrow{P}_q$ whose elements are similar, inside each one of those subsets, but dissimilar to elements of the other subsets.

In other words, what we are stating is, that for some simulations, classes of solutions should emerge from the set of all possible solutions. It is important to notice that some simulations do

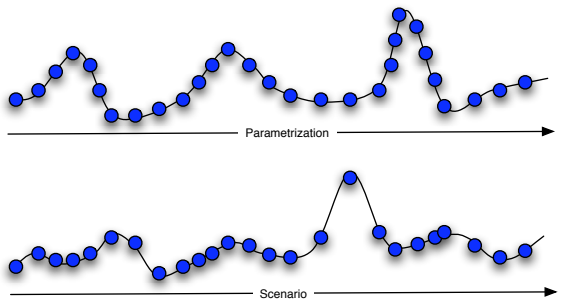

Figure 2: Continuous scenario selection

not have a clear set of different classes of solutions. Some simulations can be so erratic that similar scenarios have completely different solutions and others might have solutions so evenly scattered throughout the solution space that no classes can be identified.

4 Scenario Adaptation

One way of implementing scenario adaptation will be by simply listing all scenarios tested with their optimum parameter as calculated by the optimizer in the form:

$$S(i) = s_{i1}, \ldots, s_{in}, p_{i1}, \ldots, p_{im} \tag{1}$$

Agents could then just find the scenario from the listing that is nearest to the present simulation conditions (see Figure 2). A simple metric, like the euclidean distance, could be used to determinate the correct scenario. This approach, although simple, has some disadvantages:

- The scenario listing might be to extensive. This could make finding the best scenario computationally impossible.
- In some simulations constantly changing parameters can be complicated or undesired. This method would force simulations to change their parameters only due to small scenarios fluctuations.

The second problem could be solved by only changing the agent parameters in regular intervals (see Figure 3). This would unfortunately cause other problems:

- In simulations where conditions do not change often, at least not dramatically, but that need a swift response when they do, this method could cause a slow reaction to environmental changes.
- Even if not as regularly this method would force configuration changes even when not strictly necessary.

A different approach would be to only change the configuration when the current scenario had an optimum configuration that differed significantly from the current one (or when the current scenario changed dramatically).

The problem with this approach is how to evaluate if a configuration/scenario differs greatly from another one. With no sense of scale it becomes tricky to make any type of decision on when to change to another parameter configuration. One solution would be to first analyze, for example, average distances between elements. A better one would be to classify the solutions and group them according to their similarity.

A different method of implementing adaptiveness could be developed by using the optimum parameter and scenario classes that the aggregator module produces. This could be done, if for every scenario class we had a representative parameter configuration. This configuration could be

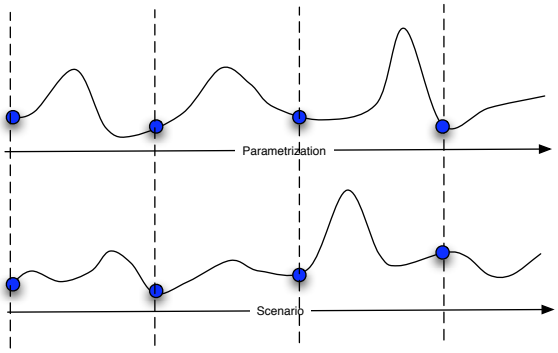

Figure 3: Time based scenario selection

for example the centroid of the configuration class or the tested configuration most near to the centroid. To use this method we would need the aggregator to output its results in the following form:

$$S(i) = sc_{i1}, \ldots, sc_{in}, p_{i1}, \ldots, p_{im} \tag{2}$$

Where sc_{ij} is the coordinate j of the centroid of scenario class i and p_{ik} is the value of the parameter k that was found to be optimum for that same scenario class.

Some different methods can be used to find which parameter configuration to use for each scenario class:

- Test all parameters configurations found for that class against all scenarios. This method might reveal impracticable due to performance reasons;
- Find the nearest tested parameter configuration to the class centroid;
- Select a sample of the parameter configurations of that class and test them against the class scenarios;
- Select random parameter configurations nearer the class centroid and test them against the class scenarios;
- Apply an optimization algorithm but this time use the average result from the complete scenario class instead of testing one scenario at a time.

The advantages of this method over the Nearest Scenario approach are that we can react to sudden changes quickly and, at the same time, we are not changing the simulation configuration constantly as a reaction to small environmental changes.

Then main concern one has to have when using the Nearest Aggregate approach, is to be sure that the parameter configuration chosen for each scenario is good enough for all the elements in that scenario. This can be easily done by testing that configuration against all representatives of the scenario class. As we do not know the scale of the simulation results, we need some kind of input from the user to be able to determine how much of a loss is admissible.

5 Test Case Scenario

In order to find a simulation that allowed the testing of all the desired aspects of the implemented system, several characteristics were sought:

- **Different Scenarios** - The simulation had to have different scenarios that required completely different parameter configurations. This would allow testing the optimization module, as well as the aggregation module;

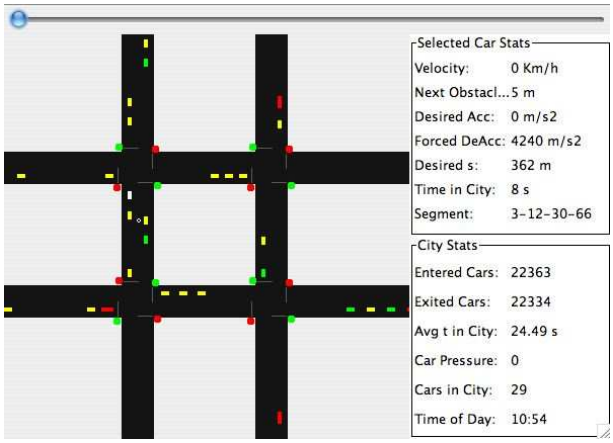

Figure 4: Traffic Simulation Interface

- **Parametrized Configurations** - The agents should have a set of configurable parameters. The optimum configuration of these parameters should depend on the current scenario;
- **Dynamic Scenarios** - It should be possible to create a dynamic scenario. In this way the simulation would experiment different parameter configuration needs during each run. This would allow testing the scenario adaptation capabilities of the system;
- **Simulation Speed** - A simulation run should not take too long in order to allow the maximum number of tests possible;
- **Stochasticity** - A stochastic simulation would allow testing if the simulation adapted well to this type of situations.

Several possible simulations were considered, and in the end the choice was to implement a very simple traffic simulation system. This simulation has all the characteristics listed previously and was fairly easy to implement. The model chosen was very simple:

- Several roads, each with only one lane in either direction;
- Roads could be either vertical or horizontal;
- Each car would enter the city in a certain lane and exit the city in that same lane. Lane changing, or turning, were not considered to keep the simulation simple;
- Traffic lights at each road intersection. A traffic light could be in one of three states: open for vertical traffic, open for horizontal traffic or changing states (yellow light);
- Cars would follow the car in their front according to a simple driver model [8].

Besides having the characteristics just listed, in this simulation it was expected that classes of solutions (containing different agent parameter configurations) would emerge.

6 Tool Implementation

A generic optimization system has been implemented to validate the ideas behind this paper. Four main points are behind the construction of this prototype:

- In the simulation optimization field, researchers often develop their own optimization systems. This happens mainly because it is relatively easy to develop a fairly decent optimizer from scratch. However having a generic optimization system available would allow the researcher to optimize his simulation with several, and perhaps more advanced, optimization methods and use some already developed analysis tools.

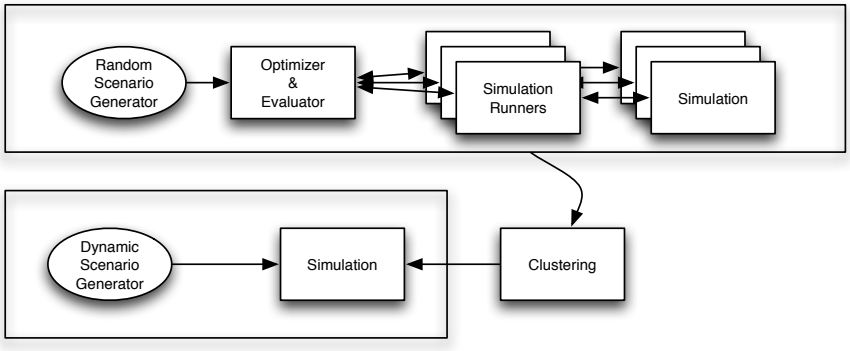

Figure 5: Generic Optimizer and Aggregator Architecture

- No system is ever generic enough for everyone. So, having the possibility of extending a system is crucial when developing a generic system. In particular this optimization system should allow new optimization methods to be added easily.
- Simulation based optimizations are always, or almost always, CPU intensive. This happens because optimization algorithms need to run simulations, which sometimes are already CPU intensive, numerous times, in order to achieve good results. An usable optimization system must take this into account.Distribution is the most obvious way of working around this problem, so a good optimization system should allow the distribution of workload across different machines.
- Simulations are used many times to optimize a set of parameters that will be later used in the real world. Most optimization systems approaches do not take into account the fact that there are environment changes occurring every time in real life applications.

As explained before, a simulation normally receives a set of parameters and outputs a set of results. Besides that, most simulations have a set of environment parameters that can be user adjusted or randomly selected.

A generic optimization system must cover as many different configurations as possible. However, creating a system that is to complicated to use should be avoided. So, a few concessions had to be made in order to keep the system simple. However, the system allows different simulation configurations to be used by means of system extensibility.

The system is composed by several modules that will be introduced in the following paragraphs and are explained in full detail in [5].

To begin, a module that will interact with the different type of simulations was needed. This module is, first of all, distributable so we can have several instances of it running in different machines. It should also be able to receive a binary file for a certain simulation, instructions on how to run it with different parameters and how to gather results from it.

A second module, that works closely with the one just described, is responsible for evaluating simulation runs. Another task of this module is to mask the fact that simulations are normally stochastic in nature.

Optimization is obviously the major goal of the system so an optimization module is essential. This module uses the evaluator module in order to get the results from various simulation runs and use these results to find the optimum parameters for a given scenario.

A final module aggregates and analyzes results from the optimizer, in order to create dynamic optimization schedules for the simulation, and allow the user to better understand how parameters and scenarios influence each others. The clustering module receives the optimum parameters for each scenario from the optimizer and outputs the dynamic optimization schedules that are then used by a simulation having dynamic scenarios. Figure 5 captures the various modules and their interaction.

Table 1: Adaptative Simulation Results

	Avg/Max Waiting Time	Max Queue Size	Configuration Changes
Single	28/87s	610 cars	0
Complete	30/94s	130 cars	1116
Clustered	23/82s	17 cars	162

7 Results

Several different scenarios were randomly generated with an optimization algorithm (in this case a Genetic Algorithm [2]) applied to each one of them. This generated approximately one hundred different scenarios and their respective agent optimum parameters.

These results were then aggregated into several representative clusters using the K-Means algorithm. This step, produced a set with only nine different scenarios.

An altered version of the traffic light simulator was developed, that instead of receiving a static traffic light schedule, would receive a set of scenario and agent parameter configuration pairs. This version of the simulator would constantly select the scenario, from that set, that most resembled the current scenario and use the traffic light configuration that was considered as the best for that particular case. This setting was run with three different configuration files:

- The complete set of scenarios (one hundred) and their optimum parameter configurations.
- A significantly smaller set of scenarios created with base in the larger set using the aggregation process.
- A single optimum parameter configuration that was calculated as being the best for an average scenario.

Table 1 captures the most important results from each test run.

Using a smaller subset of results, one would expect fewer changes in the used configuration. In fact Table 1 shows exactly that. Using the complete set of results the simulation changed parameters 1116 different times against 162 times using the clustered subset.

As less information was available, theoretically the simulation could not adapt as well as in the previous method. However, using the clustered results did not affect the simulation performance and even made it more efficient. There is a simple reason that explain this, at first glance, awkward behaviour. This happened because constantly changing parameters can lead to a loss in performance. In this particular case each time the parameters changed the timing of the traffic lights would be affected momentarily causing smaller, or longer, traffic light patterns than the optimum ones. In other scenarios the cost of changing parameters could be even higher (e.g. if the agents had to change their planning every time the parameters changed).

8 Conclusions

Agent based simulations, involving complex and dynamic scenarios, have poor results when single static configurations are used. One way of coping with this problem is to have a large set of configurations that will be used in each specific scenario. The problem is that some simulations do not cope well with configuration changes. This can be true either because changing the configuration is expensive or because a period of instability is created when the configuration is changed.

In this paper, it has been shown that using clustering algorithms to create smaller, but still meaningful, sets of configurations is a valid method that can be used to minimize the number of possible configurations thus minimizing the number of configuration changes. In the particular

simulation tested it has also been shown that using this same method, better results can be achieved. The K-Means clustering algorithm has been shown as an effective clustering algorithm for this particular problem. More complex simulations might require different clustering algorithms.

Other tactics to minimize the number of times the configuration is changed have been discussed like, for example, only changing the configuration when the current scenario differed significantly from the last scenario where the configuration has been changed. It has been explained that this alternative can create some other problems, like estimating how much change is needed for a scenario to be significantly different from another. Other problems would exist if two close scenarios needed radically different approaches. Another approach would be to analyze if a configuration change is needed from time to time. This alternative approach would not work well if the simulation was very dynamic and quick responses were needed when the scenario changes.

9 Future Work

The only clustering method implemented as support for this dissertation has been the K-Means algorithm. Further testing could be done with different clustering algorithms or with variations of the used algorithm.

In this paper it has been assumed that clusters always have a n-dimensional spherical form. This is not always true. Optimization results can form very different types of clusters like ellipsoids and stripes or even have a completely different formations that can only be defined mathematically. In the later case, clustering algorithms would no longer be useful and other methods had to be studied.

The method presented for scenario adaptation is just one of many different possible approaches. Several other methods could be researched. One obvious problem that needs to be tackled is that of simulations where it is hard to make runs in static scenarios and are difficult to assess using small time sections of the simulation. For example, in the Robo Soccer Simulated League [6], the scenario could be seen as a set of different parameters, representing things like the behavior of the other team, remaining time or ball position. In this case it would be impossible to run the simulation with a static scenario as some of these parameters would change throughout the simulation.

10 Acknowledgements

This work was partially supported by FCT Project FCT/POSC/EIA/57671/2004 (ABSES - Agent Based Simulation of Ecological Systems).

References

[1] Farhad Azadivar. A tutorial on simulation optimization. In *WSC '92: Proceedings of the 24th conference on Winter simulation*, pages 198–204, New York, NY, USA, 1992. ACM Press.

[2] Thomas Bäck and Hans-Paul Schwefel. Evolutionary computation: An overview. In T. Fukuda, T. Furuhashi, and D. B. Fogel, editors, *Proceedings of 1996 IEEE International Conference on Evolutionary Computation (ICEC '96), Nagoya*, pages 20–29, Piscataway NJ, 1996. IEEE Press.

[3] C. Fraley and A. E. Raftery. How many clusters? Which clustering method? Answers via model-based cluster analysis. *The Computer Journal*, 41(8):578–588, 1998.

[4] A. K. Jain, M. N. Murty, and P. J. Flynn. Data clustering: a review. *ACM Computing Surveys*, 31(3):264–323, 1999.

[5] André Restivo. Dynamic scenario simulation optimization. Master's thesis, Faculdade de Engenharia da Universidade do Porto, 2006.

[6] RoboCup. Robocup international homepage (http://www.robocup.org/), 2006.

[7] J. Swisher, P. Hyden, S. Jacobson, and L. Scruben. A survey of simulation optimization techniques and procedures. In K. Kang J.A. Joines, R.R. Barton and P.A. Fishwick, editors, *Proceedings of the 2000 Winter Simulation Conference*, 2000.

[8] Martin Treiber, Ansgar Hennecke, and Dirk Helbing. Congested traffic states in empirical observations and microscopic simulations. *Physical Review E*, 62:1805, 2000.