# Automatic tuning of agent-based models using genetic algorithms

Benoît Calvez, Guillaume Hutzler

# Automatic Tuning of Agent-Based Models Using Genetic Algorithms

Benoît Calvez and Guillaume Hutzler

Universite d'Evry-Val d'Essonne/CNRS
LaMI, UMR 8042
523, Place des terrasses de l'Agora 91000 Evry, France
{bcalvez,hutzler}@lami.univ-evry.fr

**Abstract.** When developing multi-agent systems (MAS) or models in the context of agent-based simulation (ABS), the tuning of the model constitutes a crucial step of the design process. Indeed, agent-based models are generally characterized by lots of parameters, which together determine the global dynamics of the system. Moreover, small changes made to a single parameter sometimes lead to a radical modification of the dynamics of the whole system. The development and the parameter setting of an agent-based model can thus become long and tedious if we have no accurate, automatic and systematic strategy to explore this parameter space.
That's the development of such a strategy that we work on suggesting the use of genetic algorithms. The idea is to capture in the fitness function the goal of the design process (efficiency for MAS that realize a given function, realism for agent-based models, etc.) and to make the model automatically evolve in that direction. However the use of genetic algorithms (GA) in the context of ABS brings specific difficulties that we develop in this article, explaining possible solutions and illustrating them on a simple and well-known model: the food-foraging by a colony of ants.

## 1 Introduction

Agent-based simulation (ABS) is interested in the modelling and the simulation of complex systems. Its aim is to reproduce the dynamics of real systems by modelling the entities as agents, whose behavior and interactions are defined. A first validation of such models is obtained by comparing the resulting dynamics, when the model is simulated, with that of the real system (measured thanks to experimental data). Similarly, Multi-Agent Systems (MAS) are designed so as to accomplish a given function in a collective and decentralized way. The validation of the system is thus given by the fact that the function is realized and that it is efficient.

In both cases, one of the crucial aspects of the design process lies in the tuning of the model. Indeed, this kind of model is generally characterized by lots of parameters which together determine the global dynamics of the system. The

search space is thus gigantic. Moreover, the behavior of these complex systems is often chaotic: on the one hand small changes made to a single parameter sometimes lead to a radical modification of the dynamics of the whole system; on the other hand some emergent phenomena are only produced in very specific conditions and won't occur if these conditions are not met. The solution space can thus be very small. As a consequence, the development and the parameter setting of an agent-based model may become long and tedious if we have no accurate, automatic and systematic strategy to explore the parameter space.

The approach that we suggest is to consider the problem of the development and the validation of ABS or MAS models as an optimization problem. The validation can thus be reformulated as the identification of a parameter set that optimizes some function. The optimization function for ABS would be the distance between the artificial model that we simulate and the real system. The optimization function for MAS would be the efficiency in the realization of the function. Given the large dimensionality of the problem, optimization techniques such as Genetic Algorithms (GA) can then be used to explore the parameter space and find the best parameter set with respect to the optimization function.

However the use of genetic algorithms in this context is not so simple. The first reason lies in the choice of the fitness function: agent-based systems or simulations are dynamic and often characterized by emergent and transitory phenomena, which complicates the measure of the fitness function. What has to be measured and when, are questions that strongly influence the characteristics of the models obtained by the algorithm, thus the quality of the results. If the fitness function is not carefully chosen, the resulting models will be optimized for that specific fitness function, which may not correspond to the initial goal of the designer of the model. The second reason is that no mathematical model allows to anticipate the dynamics of an agent-based model without executing it. The computation of the fitness function then requires the execution of the multi-agent system or simulation (and even several executions to take the stochasticity of the system into account), which implies a high computational cost. It is thus necessary to develop strategies to accelerate the convergence of the algorithm.

In section two we present the problematics related to the parameter tuning of an agent-based simulation. Then in section three we present the general framework of genetic algorithms and show the difficulties that arise from the application of these techniques to agent-based simulation. In section four, we propose guidelines for the use of genetic algorithms with agent-based simulation and show how it applies to the example of ant-foraging, before concluding in section five.

## 2 Parameter tuning

### 2.1 Parameters of agent-based models

In the context of agent-based simulation, a model and the simulator with which it is executed include lots of parameters. These parameters can be of different

natures. Some parameters are peculiar to the simulator: the discretization step for the modeling of time and space for instance can be a fixed feature of the simulator. As a consequence, these parameters can generally not be modified by the user. For this reason, we do not include this type of parameters in our parameter space. We only include the parameters that are specific to the model. Some of them can be extracted from the knowledge of the field (either experimental or theoretical) and can thus be associated to fixed values. Other parameters have to be kept variable, which can be for different reasons: on the one hand, the knowledge of the field is generally not exhaustive (which is the reason why we build a model and simulate it); on the other hand, this knowledge may not be directly compatible with the model. In this case, a common approach can be to try some values and simulate the model to see how it behaves globally. What we propose is to have a general approach to automate this process.

This problem appears to be quite common in the modelling and simulation of any kind of complex systems. This is the case especially for social simulation, as it was shown by Sallans et al. [1]. In the context of integrated markets model, they present a model with lots of parameters, whose design and validation requires specific techniques to select appropriate values for the parameters.

### 2.2 Objective

Depending on the motivation of the modeling work, the criteria used to explore the parameter space will also be different. This motivation may be to model and simulate a real system, but it can also be to study the discrete models that may produce a given emergent phenomenon. Finally, the motivation may be to propose models that perform best in the realization of a specific function.

In the first case, we want to check if the simulated model correctly grasps the behavior of the real system. The validation of the model will thus be to have a behavior identical to (as close to as possible) experimental knowledge. The search problem can be seen as the search of the parameter set that minimizes the distance between real and simulated data.

Having a similar behavior can also mean that specific emergent phenomena known to occur in a real system can be observed in the simulation. Emerging ant lines for example, will only occur if the chemical trails have specific properties, as we will see in next section. The emergence of this phenomenon will thus be associated to specific parameter values, and the search problem will consist in searching the different ranges of parameters where an emergent phenomenon is observable. In some cases, choosing slightly different values may lead to completely different results during the simulation, which complicates a manual exploration of the parameter space and justifies the development of automatic techniques.

### 2.3 Example

We will present the parameter setting of an agent-based model with the example of ant foraging (we use the multi-agent programmable modeling environment

NetLogo [2] and its "Ants" model). Figure 1 shows this example. The principle is that each ant searches for food and brings it to the nest secreting a chemical on the way back. When other ants feel the chemical, they follow the chemical way up to the food source, which reinforces the presence of the chemical and finally produces trails between the nest and the food sources. We can see ant lines emerging, which are similar to the ones that we can observe in natural conditions. In the model that we used, there is a nest in the center of the area of the simulation, and three food sources around the nest.

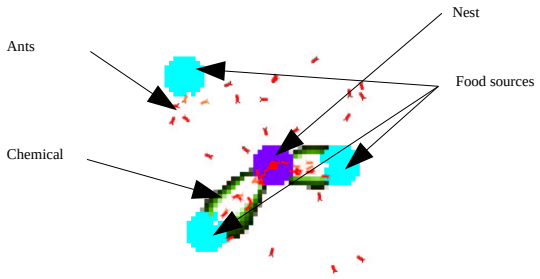


**Fig. 1.** Example of an ABS: Ant foraging

In this model, two parameters condition the formation of chemical trails. The first one is the diffusion rate of the chemical, which corresponds to the fact that a given proportion of the chemical will be diffused to the neighboring patches (regions of the environment) at the next time step. This is used to simulate the diffusion of the chemical in the atmosphere. The second parameter is the evaporation rate of the chemical, which corresponds to the fact that a given proportion of the chemical will disappear from the patch at the next time step. This is used to simulate the evaporation of the chemical in the atmosphere.

If we change the second parameter, we can get different behaviors of the simulation. Table 1 shows the variation in the evaporation rate.

| | Model 1 | Model 2 | Model 3 |
|---|---|---|---|
| Diffusion rate | 50 | 50 | 50 |
| Evaporation rate | 0 | 15 | 99 |

**Table 1.** Models with different evaporation rates.

Figure 2 shows the results during the simulation. We get two different global dynamics for the system. The first dynamics (simulation 1 and 3) is a random food search (either because there is too much chemical or because there is none). The second dynamics (simulation 2) is a food search with ant lines.


**Fig. 2.** Simulation results for the three models of table 1

For example, we can be interested more precisely in the dynamics of ant lines. Table 2 shows three models with small modifications for the two parameters. We can see in figure 3 that these small variations may lead to different dynamics: the difference lies in the way that food sources are exploited. In model 1, food sources are exploited in turn while in model 3, they are all exploited at the same time. As a result, we observe one, two or three ants lines.

| | Model 1 | Model 2 | Model 3 |
|---|---|---|---|
| Diffusion rate | 40 | 50 | 60 |
| Evaporation rate | 15 | 15 | 20 |

**Table 2.** Models with slightly different parameters.

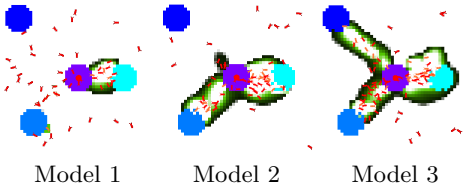


**Fig. 3.** Simulation results for the three models of table 2

### 2.4 Previous work

Different methods have already been proposed to explore automatically the parameter space of discrete models. In the `NetLogo` platform for instance, the "BehaviorSpace" [2] tool allows to explore automatically and systematically the parameter space. This space is a Cartesian product of values that each parameter can take, some of them being chosen as fixed, others being limited to a subset of

all possible values. However when we have lots of parameters, some of which can take a good many values (real-valued parameters for example), the parameter space becomes huge and the systematic exploration becomes impossible.

Other methods have been proposed, which differentially explore the whole parameter space, focusing on the most interesting areas. That's the case of the method developed by Brueckner and Parunak [3]. They use a "parameter sweep infrastructure", which is similar to the "BehaviorSpace" tool of `NetLogo` . However, to avoid a systematic exploration, they use searcher agents and introduce the fitness notion. The aim of a searcher agent is to travel in the parameter space to look for the highest fitness. Starting from a given location in the parameter space, searcher agents have two choices: move or simulate. Each agent chooses according to the confidence of the fitness estimate (proportional to the number of simulations at this point) and the value of the fitness. If it chooses to move, it heads for the neighboring region with highest fitness. A disadvantage of this method is that searcher agents may head for local fitness maxima.

Another method is to add knowledge to the agent-based model, as is the case with white box calibration [4]. The principle is to use the knowledge of the agent-based model to improve the tuning process. The aim is to reduce the parameter space by breaking down the model into smaller submodels, which can be done using different methods (General Model Decomposition, Functional Decomposition, ...). Each of the submodels is then calibrated, before merging them back to form the model. The division and fusion operations are the difficult steps of the method. The division operation, on the one hand, requires the addition of knowledge about the model, which may not be available. The fusion operation, on the other hand, has to merge calibrated submodels into a calibrated higher model, which is not automatic.

As we saw previously, Sallans et al. [1] have lot of parameters in their model. Some of these parameters are chosen based on initial trial simulations. The other parameters are chosen by the Metropolis algorithm, which is an adaptation of the Markov chain Monte Carlo sampling to do a directed random walk through parameter space. This method performs well on a continuous parameter space, but will hardly be usable when the parameter space is chaotic and there is the stochasticity in the simulation.

## 3 Use of genetic algorithms

As the tuning of the parameters of a model is a strongly combinatorial problem, we propose to use genetic algorithms, which generally provide good results on problems of this kind.

### 3.1 Principe of genetic algorithms

Genetic algorithms are a family of computational models inspired by evolution. They allow to solve various classes of problems, more specially optimization problems. Figure 4 shows the classic schema of genetic algorithms. In this framework,

the potential solution of a problem is encoded on a linear data structure, which is called a chromosome. The algorithm works on a set of several chromosomes that is called a population. Operators are applied to this population.

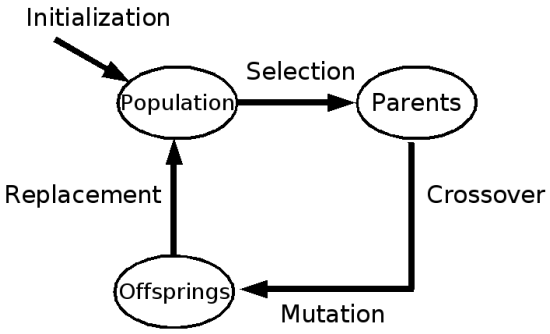


**Fig. 4.** Diagram of a standard genetic algorithm

The population of chromosomes is initialized randomly. Each chromosome is then evaluated using a fitness function, which measures how good this potential solution is with respect to the initial problem: it comes to give a score to each chromosome.

A selection is made among the population of chromosomes: we obtain a new population named parent population. Recombination and mutation operators are then applied to this population: we obtain a new population named intermediate population. The recombination consists in swapping parts between two chromosomes. With this operation, we obtain two new chromosomes. That's the most frequent operator in genetic algorithms. Intuitively the role of this operator is to pick up the best part of chromosomes to obtain a better chromosome. The mutation consists in changing a part of a chromosome. This operation avoids converging prematurely to a local solution.

The new chromosomes of the intermediate population are evaluated. A new population is finally created from the initial population and the intermediate population, before starting again the whole process.

### 3.2 Choice of the fitness function

If we consider the exploration of the parameter space as an optimization problem, we need to define very carefully the function that will have to be maximized (or minimized) by the algorithm. This fitness function is of fundamental importance since the models that will be selected are the one that perform best with respect to this function. In the context of agent-based simulation, the choice of the fitness function is problematic for several reasons: as a first thing, it is not the result of a computation but the dynamics of a process that has to be assessed; secondly,

emergent phenomena may be difficult to characterize quantitatively since they are often related to a subjective interpretation by a human observer.

**Quantitative vs. qualitative.** Validating an agent-based model by assessing the distance between the simulation and the real system can be done either quantitatively or qualitatively.

In the quantitative case, data are measured in the simulation and compared to data measured in similar conditions in the real system. The distance between the simulation and the real system is then the Euclidean distance between the two data vectors. If we try to select models that are optimized for the realization of some function, the fitness function can also be directly measured by the performance of the system for that function. In the case of ant foraging, this would correspond to the quantity of food retrieved to the nest after a given period or the time necessary to bring all the food back to the nest.

In the qualitative case, what is important is that a given emergent phenomenon be present in the simulation. The difficulty is then to translate this observation into a quantitative measure (the fitness function). In the example of ant foraging, we know from the observation of real ants that they organize dynamically along lines between the nest and food sources because of the creation of corresponding chemical trails. The fitness function could then be designed so as to reward models in which complete chemical trails are formed between the nest and the food sources. In some cases, the characterization of such emergent phenomena may not be so simple since it may be the result of a subjective interpretation by an observer, which cannot be captured easily by a quantitative measure.

**A dynamic process.** In classical optimization problem, the fitness function corresponds to the result of a computation. Therefore, the question of the time at which the measure should be made doesn't make sense: the measure is done when the computation has ended. On the contrary, agent-based simulations are dynamic processes that evolve along time and generally never end.

We can clearly see in the examples given in the previous section that the evaluation of the fitness function generally has to be done at a given time-step of the simulation. The choice of this time-step is not neutral and may greatly influence the performance of the genetic algorithm and the resulting model. If we try for example to select models that exhibit a given behavior that is transitory, it may not be sensible at the time-step chosen for the evaluation of the fitness function and the measure should thus be repeated at different time-steps.

**Example** We can show with an example the difficulty of the choice of the fitness function. Figure 5 shows the foraging simulation at five different time-steps.

If we choose a quantitative fitness (quantity of food brought back to the nest after a given period) and if the computation of fitness function is too early ($t \leq 30$) or too late ($t \geq 600$), the value of the fitness will be the same regardless

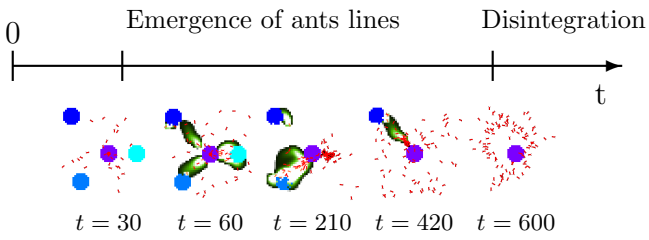


**Fig. 5.** Ant foraging at different time-steps

of the model: if the measure is too early, no food at all has been brought back, even in the best model; if the measure is too late, all the food has been brought back, even in the worst model. We have thus to choose the "right" moment to evaluate the fitness function (in this case between 60 and 420). However, this moment may not be the same for all different models, which suggest that it may be useful to evaluate the fitness at different time-steps.

If we choose a qualitative fitness (existence of ant lines) and if the computation of the fitness function is too early ($t \leq 30$) or too late ($t \geq 600$), we see no ant lines. Again, we have to choose the "right" moment to evaluate the fitness function (in this case between 60 and 420). But the number of ant lines also evolves during the simulation, which suggest again that the fitness should be evaluated at different time-steps on this interval.

### 3.3 Computation of the fitness function

**Time.** Since no mathematical model can anticipate the dynamics of an agent-based model without executing it, the computation of the fitness function requires one or even several simulations. This means that the time required to compute the fitness function will be significant (which is generally not the case for classical optimization problems). The global computation time to run the genetic algorithm is $(n \times N) \times T_f$ where $n$ is the number of chromosomes, $N$ is the number of generations, and $T_f$ is the time to compute the fitness function. If $T_f$ is 10 minutes, $n$ is 20 chromosomes and $N$ is 100 generations, then the global time to run the genetic algorithm is nearly two weeks. We must therefore find methods to reduce either the number of chromosomes, the time to converge towards an optimum or the time to compute the fitness function. We mainly studied the last possibility through distributed computation and fitness approximation[5].

*Distributed computation.* Since the different models are independent from each other, the evaluation of their fitness is also independent. Therefore each evaluation of the fitness (that is to say each agent-based simulation) can be done on a different computer. We can thus have several computers to simulate the models and use the master-slave parallel genetic algorithms [6], which improves the performance as compared to standard GA.

*Fitness approximation.* Fitness approximation comes to approximate the result of the simulation by a mathematical model, such as a neural network or a polynomial for instance. We tried this approach by training a neural network with test data. After the learning phase, we used it to compute the fitness, with the generation-based control approach, in which the whole population during $\eta$ generations is evaluated with the real fitness function in every $\lambda$ generations [7]. The results however were not so good and this approach has been temporarily abandoned. We suspect in that case that the approximation was not good enough to obtain satisfying results but this has to be explored in more details.

**Stochasticity.** Two agent-based simulations can generally bring slightly different results, even if the underlying model is exactly the same, due to the stochasticity of the model and of the simulator. One simulation is not enough to evaluate the fitness function: it can only be considered as an estimate for the fitness.

We studied the stochasticity of the "Ants" model and the `NetLogo` simulator. We chose three different fitness functions :

- the first fitness function is the quantity of food brought back between 100 and 200 simulation time-steps;
- the second fitness function is the time to bring back all the food to the nest;
- the third fitness function is the number of ant lines.

We assessed, depending on the number of simulations, the error rate in the estimation of the fitness as compared to the "real" fitness (estimated with 100 simulations). The results are shown in table 3.

| | Fitness 1 | Fitness 2 | Fitness 3 |
|---|---|---|---|
| Number of Simulations | Error rate | Error rate | Error rate |
| 1 | $\simeq$34.57% | $\simeq$10.92% | $\simeq$13.28% |
| 5 | $\simeq$14.74% | $\simeq$4.85% | $\simeq$6.34% |
| 10 | $\simeq$10.3% | $\simeq$3.38% | $\simeq$4.39% |
| 15 | $\simeq$8.3% | $\simeq$2.85% | $\simeq$3.58% |
| 20 | $\simeq$7.26% | $\simeq$2.39% | $\simeq$3.15% |
| $\sigma^*$ | 0.41 | 0.14 | 0.17 |

**Table 3.** Example of the stochasticity

We see, for instance, that a 5% error on the estimation of the fitness 2 can only be obtained by simulating the model more than five or ten times. But the stochasticity is more or less important depending on the fitness. Fitness 1 for example is much more sensitive than fitness 2.

In such noisy environments, a first solution is to increase the size of the population [8,9]. To multiply the number of the simulated models reduces the

effect of the stochasticity. A second solution is to simulate each model several times to improve the evaluation of the fitness function. Both solutions greatly increase the number of simulations, thus the time, of the genetic algorithm.

Another solution is to use the same technique as with fitness approximation. A solution to the stochasticity problem is then to estimate the fitness of each model with one simulation, and each $n$ generations of the GA ($n$ to choose according to the stochasticity of the model and the desired quality of the estimation), to estimate the fitness of each model with $x$ simulations. Figure 6 shows the general framework to take stochasticity into account.

We use the elitism genetic algorithm [10] that is to say we keep the best chromosomes during the algorithm, which allows to continuously improve the solution. Our implemented genetic algorithm replaces only 25 % of the population at each generation. Every 3 generations, we estimate the fitness of the models with more simulations.

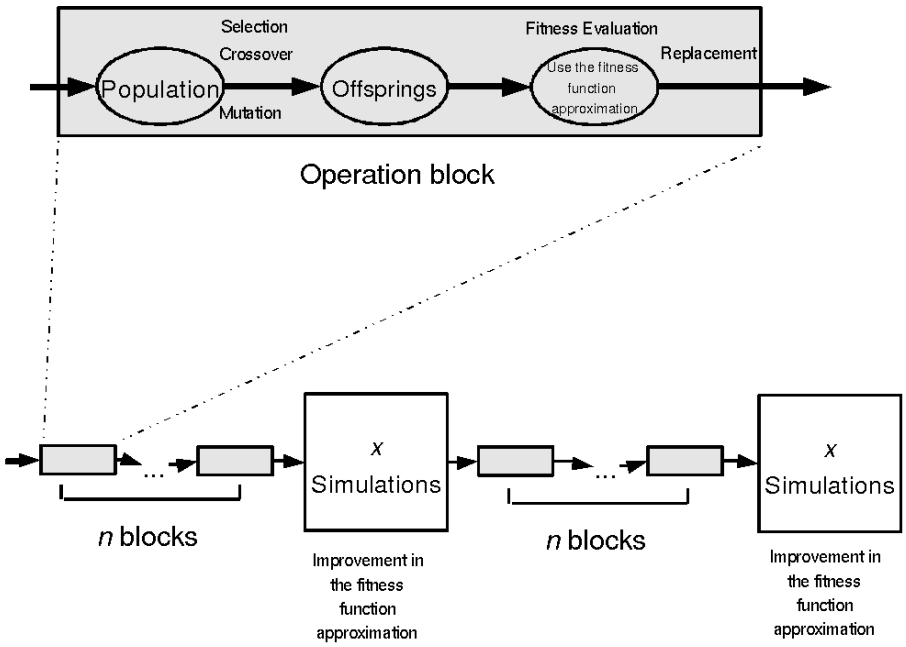


**Fig. 6.** Framework with the stochasticity

## 4 General framework and application

Up to now, we identified several difficulties peculiar to agent-based systems for the application of genetic algorithms. We now propose a general framework or

guidelines for the application of GA in this very specific context and show with several examples how it may be used.

1. determine the goal of the study;
2. elaborate the agent-based model;
3. choose the parameters of the model to evolve by genetic algorithms;
4. choose the fitness function : what do we want to optimize; when and how should we evaluate the function;
5. study the stochasticity of the model; simulate the model several times and study the results (calculate the standard deviation); determine the procedure for the exploration accordingly;
6. study the computation time of the simulation; If the simulation requires much time, use distributed computation; If the simulation requires too much time, use fitness approximation;
7. choose the number of chromosomes in the population;
8. run the genetic algorithm.

We will now detail how this may apply to the ant foraging example. In the three first examples, we use the model that has already been presented in the previous sections, with 10 ants. The main difference between the three first examples is related to the fitness function.

### 4.1 Example 1

*Experience.*

1. our goal is to optimize the foraging behavior;
2. the model is `NetLogo` "Ants" model;
3. the parameters that we evolve are the diffusion rate and the evaporation rate;
4. the fitness function is the quantity of food brought back between 100 and 200 simulation time-steps
5. The result of the study of the stochasticity is shown in table 3 for fitness 1; to evaluate the fitness function we use one simulation; every 3 generations we use 10 simulations to evaluate the fitness function;
6. the evaluation of the fitness function (that is to say a simulation) requires about 15 seconds; we use only one computer; one night of calculation is enough to compute 100 generations;
7. we take 20 chromosomes;
8. we run the genetic algorithms for 100 generations.

*Results.* We execute one run of the genetic algorithm. Figure 7 shows the results. The curve depicts the fitness of the best chromosome according to generations. And the crosses show the fitness of the best chromosome, computed by 10 simulations every 3 generations. The instability of this curve shows the stochasticity of the simulator and its model.

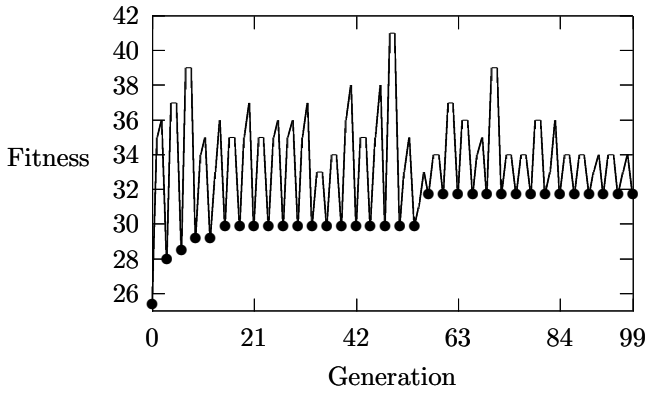


**Fig. 7.** Result of the genetic algorithm

At the end of the 100 simulation steps, the ant lines are built. During the next 100 simulation steps, the ants exploit the food sources using the chemical trails. The best models are the ones where the three food sources are exploited at the same time. However the evaporation rate is rather weak. (the evaporation rate is 8.1% and the diffusion rate is 88.6%) As a result, when there is no more food in a source, the chemical trail remains in the environment and the ants take time to exploit another source.

### 4.2 Example 2

*Experience.* The differences with example 1 are:

4. the fitness function is the time to bring all the food back to the nest;
5. the result of the study of the stochasticity is shown in table 3 for fitness 2;
6. the evaluation of the fitness function requires about 20 seconds for a good model and up to some minutes for a very bad model; we still use only one computer during one night.

*Results.* Figure 8 shows the results. Like the previous example, the best models are the ones where the three food sources are exploited at the same time. But the evaporation rates in this example are larger than in the previous example. It improves the dynamic behavior of ants: when a food source becomes exhausted, the ants quickly stop going to this source and search for other food sources.

### 4.3 Example 3 - Qualitative fitness

*Experience.* The differences with example 1 are:

1. our goal is to obtain models in which ant lines can emerge;

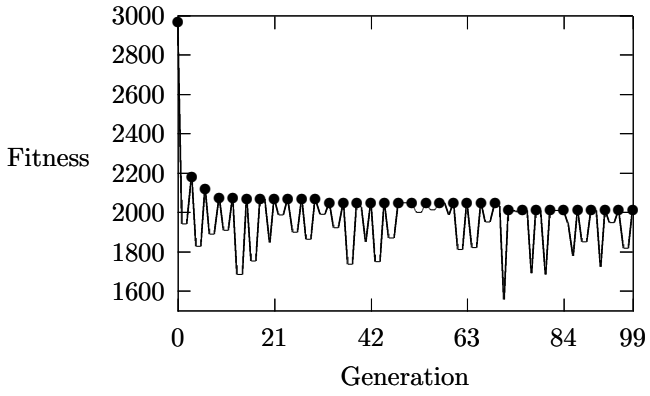


**Fig. 8.** Result of the genetic algorithm for example 2

4. the fitness is the number of ant lines; to simplify, we determine the number of continuous chemical trails between the nest and food sources; the number of trails is evaluated every 10 time-steps and the fitness is the sum of these values during 400 time-steps;
5. the result of the study of the stochasticity is shown in table 3 for fitness 3;
6. the evaluation of the fitness function requires about 30 seconds; we still use only one computer during one night.

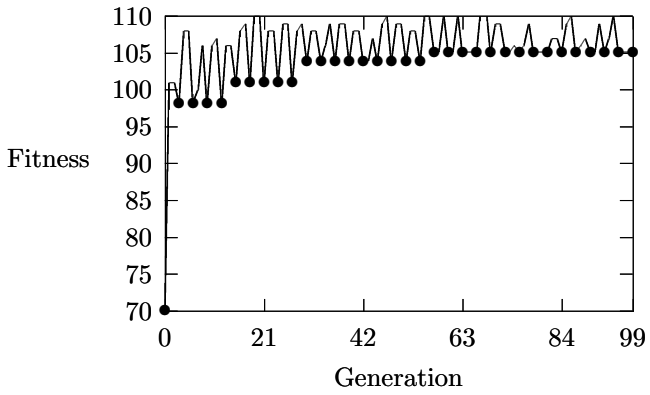


**Fig. 9.** Result of the genetic algorithm for example 3

*Results.* Figure 9 shows the results. The best models still exploit three food sources at the same time. Unlike the two previous examples, the evaporation and diffusion rates are but very small. This weakness allows to concentrate the

chemical on narrow paths without covering all the environment, but it also reduces the flexibility of the behavior of the colony of ants.

### 4.4 Example 4

*Experience.* This example is different from the three previous examples. We try to tune the model parameters in relation with the experimental data. Firstly we generated data. To this end, we use the `NetLogo` "Ants" model, with the diffusion rate set to 50 and the evaporation rate set to 15. This model is considered as the initial model, which we simulate several times (1000 simulations) during 2500 simulation steps. Every simulation step, we record the quantity of food taken back to the nest. From the different simulations, we compute the average quantity of the food back to the nest at every step: we obtain a vector of the average food quantity of the initial model.

So, the differences with the example 2 are:

1. our goal is to obtain models as close to the data as possible;
2. the fitness function is the distance between the food quantity of the simulation and the vector of the average food quantity of the initial model.

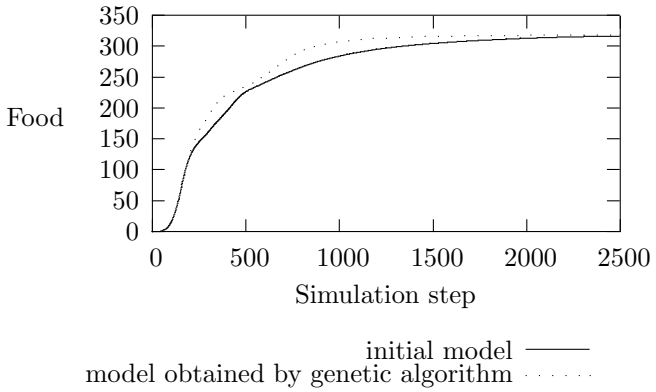


**Fig. 10.** Result of the genetic algorithm for example 4

*Result.* We execute several runs of the genetic algorithm. Figure 10 shows the results. The curves depicts the quantity of the food back to the nest every step of the best chromosome, and the average food quantity of the initial model. We can see that the two curves are very close, which indicates that the two models have a similar behavior. The parameter sets of these two models are but very different. The diffusion and evaporation rate of the model obtained by the genetic algorithm are respectively 10.86 and 20.33, and several executions of the genetic algorithm give very similar results. This indicate first that this example

is not very sensible to the parameter setting, since different settings produce similar results. Why then does algorithm converges towards a specific setting? In this case, it appears that the algorithm converges towards parameter settings that are less sensitive to the stochasticity of the model. This may be explained by the fact that the fitness function is computed most of the time with only one execution of the simulation, thus favoring the models that are close to the reference model most of the time.

### 4.5 Example 5

*Experience.* We saw in the previous example that the reference model was not very sensitive to the parameter setting. In order to make it more sensitive, we increased in this example the size of the environment with a surface area multiplied by 4. We this increase the constraints imposed to the agents in the model.

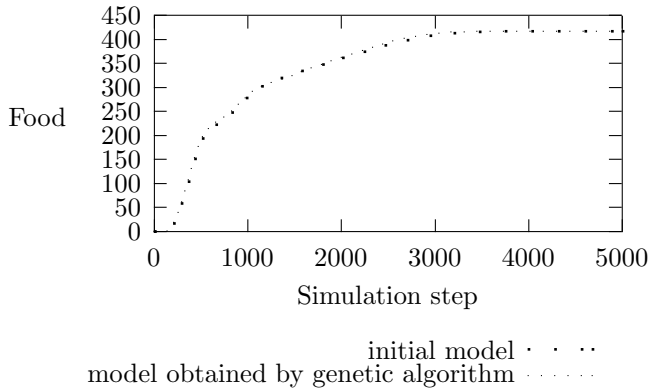


**Fig. 11.** Result of the genetic algorithm for example 5

*Result.* Figure 11 shows the results obtained with the genetic algorithm on this example. The curves depicts the quantity of the food back to the nest every step of the best chromosome, and the average food quantity of the initial model. Again, the two curves are very close. But this time, the parameter setting obtained by genetic algorithm is very similar to the parameter setting of the initial model.

### 4.6 Discussion

As we could already see with the study of the stochasticity, we obtain very different results depending on the choice of the fitness function. The models are strongly optimized for a specific fitness function and may not perform so

well with another one. The optimization creates a loss of the flexibility of the dynamics of the agent-based model. A possible solution would be to use several different initial conditions to evaluate the fitness function. We could imagine in our example to vary the position and distance of the food sources, or to add new sources dynamically to select models that show high adaptation capabilities. This would however increase again the time necessary to run the algorithm.

The optimization by the genetic algorithm also depends on the constraints imposed to the agents in the model. If a model has lots of constraints (fewer resources for example), it is necessary that it optimizes its global functioning. On the contrary, if the resources are abundant, the pressure on the model to adapt and optimize its functioning will be weaker. As a result, the use of our approach will be mostly beneficial when constraints on the model are high. In the case of ant foraging, if the fitness is the time to bring all the food back, an important resource corresponds to the number of ants. The fewer the ants, the better the organization they will need to create so as to forage efficiently. On the contrary, if the ants are very numerous, the model will perform well whatever the chemical signalling. In some cases, it may thus be useful to strengthen the constraints on the model to obtain bigger improvements.

## 5 Conclusion

This paper presents a method, based on genetic algorithms, to explore automatically the parameter space of agent-based models. We explained the specific difficulties related to the use of this approach with agent-based models: the choice of the fitness function, the stochasticity, the computational cost. We then show some possible solutions. We finally suggest guidelines to help using genetic algorithms in this context.

Then we apply the method to some simple examples: the ant foraging with different fitness functions (both quantitative and qualitative). We obtain models that are optimized with respect to a given fitness function, which is chosen in relation with specific modeling goals.

The next step is to apply the method to a more complex example. We began a work for the simulation of the glycolysis and the phosphotranferase systems in Escherichia coli. In this work, we are interested in testing the hypothesis of hyperstructures [11]. The hyperstructures are dynamic molecular complexes, enzyme complexes in the case of this work. These complexes allow to improve the behavior of a cell : more flexibility, quicker adaptation. In our study, we have 25 kinds of molecules (or agents). There are altogether about 2200 agents in the simulation. We want to study the potential interest of hyperstructures for the cell. To do this we make the rates of enzymes association and dissociation variable. In this context, the simulation of a model lasts about 10 minutes, which imposes to use the methods described in this article like the distributed computation.

To explore this complex example, we will need to develop additional strategies to reduce the parameter space (e.g. by introducing coupling between pa-

rameters), to accelerate the evaluation of the fitness function (e.g. by developing approximation methods), and to accelerate the convergence of the algorithm (e.g. by using interactive evolutionary computation[12]). Finally, another important perspective is to explore the effect of varying dynamically the simulation conditions so as to produce more versatile models.

## References


1. Sallans, B., Pfister, A., Karatzoglou, A., Dorffner, G.: Simulation and validation of an integrated markets model. J. Artificial Societies and Social Simulation **6**(4) (2003)
2. Tisue, S., Wilensky, U.: Netlogo: Design and Implementation of a Multi-Agent Modeling Environment. Proceedings of Agent 2004 (2004)
3. Brueckner, S., Parunak, H.V.D.: Resource-Aware Exploration of the Emergent Dynamics of Simulated Systems. AAMAS 2003 (2003) 781–788
4. Fehler, M., Klügl, F., Puppe, F.: Techniques for analysis and calibration of multi-agent simulations. In Gleizes, M.P., Omicini, A., Zambonelli, F., eds.: ESAW. Volume 3451 of Lecture Notes in Computer Science., Springer (2004) 305–321
5. Jin, Y.: A comprehensive survey of fitness approximation in evolutionary computation. Soft Computing Journal (2003)
6. Cant-Paz, E., Goldberg, D.E.: Efficient parallel genetic algorithms: theory and practice. Computer Methods in Applied Mechanics and Engineering **186** (2000)
7. Jin, Y., Olhofer, M., Sendhoff, B.: A Framework for Evolutionary Optimization with Approximate Fitness Functions. IEEE Transactions on Evolutionary Computation **6**(5) (2002) 481–494
8. Beyer, H.G.: Evolutionary Algorithms in Noisy Environments: Theoretical Issues and Guidelines for Practice. Computer methods in applied mechanics and engineering **186** (2000)
9. Goldberg, D.E., Deb, K., Clark, J.H.: Genetic algorithms, noise, and the sizing of populations. Complex System **6** (1992)
10. Beker, T., Hadany, L.: Noise and elitism in evolutionary computation. In: Soft Computing Systems - Design, Management and Applications. (2002) 193–203
11. Amar, P., Bernot, G., Norris, V.: Modelling and Simulation of Large Assemblies of Proteins. Proceedings of the Dieppe spring school on Modelling and simulation of biological processes in the context of genomics (2002) 36–42
12. Takagi, H.: Interactive evolutionary computation: fusion of the capabilities of EC optimization and human evaluation. In: Proceedings of the IEEE. Volume 89., IEEE Press (2001) 1275–1296