INTERACTIVE DYNAMIC PRODUCTION BY GENETIC ALGORITHMS

M.Baioletti, A.Milani, V.Poggioni and S.Suriani
Mathematics and Computer Science Department
University of Perugia
Via Vanvitelli 1, 06123 Perugia, Italy
email: {baioletti,milani,poggioni,suriani}@dipmat.unipg.it

ABSTRACT
In this work we introduce an adaptive genetic algorithm for solving a class of interactive production problems in a dynamical environment. In the interactive production problem, a system continuously generates product instances which should meet the requirements of a market of customers/agents which are unknown to it. The only way for the system to know the evaluation of a product instance is the feedback obtained after delivering it to the customer. In a dynamical environment the domain of the products is changing and the customer/agents are changing their preferences over the time. This scenario is common to many IT services and products which are continuously delivered to a mass of anonymous users. The proposed algorithm employs typical genetic operators in order to optimize the product delivered and to adapt it to the environment feedback and evolution. Differently from classical GA the goal of such system is to maximize the average result instead of determining the best optimal solution. Experimental results are promising and show interesting properties of the adaptive behavior of GA techniques.

KEY WORDS
Genetic algorithms; Adaptative systems; Interactive production problem

1 Introduction
The increasing diffusion of mass services based on new information technologies poses new requirements and goals on adaptive systems. In particular these systems have to provide adaptive personalized services or products to a mass of anonymous clients [1]. There already exists a growing variety of purely digital products which are produced and delivered instantaneously on demand to thousands of users. The front page of an online newspaper, a banner advertising a special offer, or a voice system proposing valuable offers to mobile phone customers are all example of products which are submitted to a mass of individual users, which in a short time decide to browse, to buy or to accept the proposed product. It is worth noticing that another feature of these purely digital products is their short-lived nature: news, available offers and customer preferences rapidly change over time, thus they require a quick adaptation of the product to the changing conditions of the market environment. In this scenario an adaptive system should exploit the ability of giving each user a different instance of the product, and the ability of monitoring the single customer response to the delivered product, in order to obtain the adaptation of the delivered products to the real market and a rapid adaptation of the products to the dynamical evolution of the market. Adaptive systems should, in other words, exploit the interactive aspect of the markets to drive the production.

Classical knowledge based systems improve the customer satisfaction by using the idea of matching a user model [2] against a model of the product in order to anticipate the expected user reaction. These approaches are often hard to apply to the described scenario for a number of reasons. In many cases technological (e.g. short and anonymous internet connections) or privacy reasons do not allow to build a significant user model. In other cases, a model of the many services or products does not exist, and since services and products can appear and disappear very quickly, it is not worth wasting the effort of building accurate models.

An interactive approach where genetic algorithms [3,4] interact with the external world in order to optimize a behavior has been proposed in several works, but in different scenarios. Some works (for instance [5,6,7]) have introduced “real world” issues into the GA loop. For instance [8] proposes an interactive approach where the user provides the fitness function, while in [9,10] GA based machine learning systems use historical data as fitness function.

We can point out that most GA approaches to adaptation basically operate offline, with a first phase in which they evolve the solution in a “virtual training environment”, even if the fitness comes from interaction with an external source, and then they return the best solution selected among all the generations built during the “training phase”.

In the interactive production scenario previously depicted, we have instead additional requirements. First of all, there cannot be an offline training phase because the production is a continuous online cyclic process and the solutions/products are immediately delivered to the customer. Moreover, the optimal solution itself is evolving over the time, since we assume a dynamical evolution of the domain. Under these hypotheses it is apparent that the goal of such an adaptive system is not just to find an “optimal solution”, but to optimize the score of the global customer
satisfaction.

In this paper we propose an approach for the problem of generating products in a dynamically evolving environment that is both interactive and adaptive [11], and combines communication with the external world with the typical adaptive behavior of GAs.

GA concepts, such as crossover and mutation, are powerful tools which allow to perform fast hill climbing of local minimum and plateau in optimization problems [4]. The idea of bringing these adaptive features in the scenario of interactive production is made more challenging from the facts that the population of clients asking for services/products is evolving over time and the client responses can vary as well.

The goal of the presented system is to be able to adapt the product instances when the domain change, i.e. product components, clients and clients’ opinion change over time in unknown and unpredictable way. The basic idea is to evolve populations of products by using the application world as a fitness function, under the principle the real world is the fitness [11]. The fitness function is unknown to the systems which receives some information from the external world in terms of a judgment/evaluation of the clients about the delivered products.

The interactive production scenario is intuitively and well represented by an online newspaper, where the clients give an opinion about the newspaper front page. An online newspaper also represents an highly dynamical domain. In the real world, the readers of an online newspaper change very quickly because at every instant new connections are activated and many others are closed. The publisher cannot have a user model for each client, so the users and, especially, the user preferences are unknown. The available news change quickly according to what happens in the world, and the readers interest about news can change during the time because of many unpredictable reasons. Using the GA terminology, the published news and the front pages respectively represent the genes and the chromosomes of the population; the reader’s opinion, which is external to the system, is the fitness function.

At each instant the population of front pages is proposed to the readers that gives their opinion (for example by clicking the preferred news or spending time in reading). The system should react both to the latest news updates and to the changes in the reader’s opinion, while maintaining a high rate of customer satisfaction.

In the following paragraphs the class of dynamical production problems on which we focus is described and the adaptive interactive genetic algorithm for dynamical environments is depicted. The algorithm has been tested using a simulation module; the experimental results, which shows a promising adaptive behaviour, are presented and discussed. Relevant properties which emerge from the tests are finally illustrated together with conclusions and some lines of future works.

2 The Interactive Production Problem

The Interactive Production (IP) problem, described here, consists in building good products, each of them is composed by $m$ components. There exist $N$ different components, which are chosen from a fixed repository, containing an unbounded number of copies for each component. The product evaluation is made by interacting with the external agents to which the products are delivered. The IP process consists in a continuous production/evaluation loop which lasts for a fixed period of time, whose length is described by the parameter $T$.

For sake of simplicity all the components are considered to be homogenous and interchangeable each other, although they give a different contribution to the evaluation of the products. At the same time all the products are described as unordered sets of a fixed number $m$ of distinct components.

2.1 Unknown External Fitness

The products are evaluated by some external agents, all of them give the same product the same evaluation at given time. We can then think that all the agents belong to a unique class of agents and the evaluation can equivalently be done by a single representative of this class. The agents evaluate a product $p$ with a satisfaction degree $d(p)$ computed by

$$d(p) = \sum_{i=1}^{m} d(c_i)$$

i.e. the sum of satisfaction degrees of all the components $c_i, i = 1, \ldots, m$, of $p$. Therefore a relevant feature of our IP problem is that the product evaluation is additive with respect to the single components of each product.

The genetic algorithm for the IP problem described in the next section builds and grows up a population of a fixed number of $n$ products $\{p_1, \ldots, p_n\}$ by alternating, for each time–step (generation), an evaluation phase, in which the agents evaluate the products in the population, and an evolution phase, in which the population is updated by means of genetic operators.

Note that the algorithm knows how good are the products only at the evaluation phase by taking into account the satisfaction degrees expressed by the external agents on the delivered products. It is impossible for the system to compute the satisfaction degree of a product which is not delivered, or to know the satisfaction degree of a certain component. In other terms, the degrees of the products are known only by means of a costly interaction with the agents, in which they reveal their preferences directly about products and indirectly about components, but the system is not able to evaluate by itself a new product $q$. The only thing it can do is to estimate $d(q)$ by taking into account, for instance, the satisfaction degrees of some similar products which have been revealed in an evaluation phase.

The satisfaction degree could be regarded as an “unknown external fitness” since it relies on the external agents.
and it is unknown to the system. This feature is a major departure from a traditional genetic algorithm, in which the fitness function is given as an input to the system.

2.2 Total Satisfaction Degree

The main goal of the problem is to design a system such that the overall sum of the satisfaction degrees of all the products during all the period $T$ is maximized. If $p_{ij}$ is the $i$–th product at the $j$–th generation, the objective is

$$\max \sum_{j=1}^{T} \sum_{i=1}^{m} d(p_{ij})$$

In an equivalent form, the goal can be seen as maximizing the mean computed during all the period of the average satisfaction degrees of the population.

The purpose is then to have good products as early as possible. In fact the objective is not to find optimal or near optimal products at the end of the period, but to accumulate good scores from the agents during all the period or, at least, during most of the time, except perhaps an initial "startup" period. In other terms, in usual genetic algorithms only the final point must be good, while in our system the entire trajectory must be satisfactory.

2.3 Static and Dynamic Interactive Production

In a first version of the Interactive Production problem, called static (SIP), the value $d(c_i)$ are fixed during the period $T$. This problem would be easily solved if the system could access directly to the satisfaction degrees of each components: it would be sufficient to build products with components having the highest degrees. But since this information is not available, the static problem is not so easy to solve.

In a second version, called dynamic (DIP), the value of $d(c_i)$ are affected by changes which can happen at a some time rate. These changes can model, for instance, change in the agents opinions or evolution of the domain. This problem is clearly harder than the former, because after the satisfaction degree of a component changes, a good product can become a bad product, wasting the selection and the evolution made by the genetic phases. The speed of the reaction to these changes, i.e. the adaptation ability, is then an important issue of an algorithm to solve the dynamic version of the problem.

3 The algorithm

The algorithm we have designed for SIP and DIP problems is based on a standard evolutionary scheme driven by crossover and mutation operators. An important feature to point out is that the evolution phase is interleaved with an evaluation phase which is interactive, i.e. it relies on external evaluator agents, which provide the fitness to the algorithm.

An instance of the problem is characterized by a pair $(m, N)$, where a product is made of $m$ components chosen in a repository of $N$ possible values. The algorithm is also parametrized by a triple $(n, p_c, p_m)$ which define respectively the population size, i.e. the number of product produced at each generation, and the probabilities of crossover and mutation operators.

The representation of components and products is straightforward. The components, i.e. the domain set for genes, are identified by unique identifiers in $\{1, \ldots, N\}$ and the products, which correspond to the chromosomes of the population, are represented as sequences of $m$ component identifiers, with no replications.

The evaluation phase is realized by a procedure, external to the system, which returns for each product $p_i$ in the population, the values of $d(p_i)$. These values are used as the fitness of the chromosome. In fact, as already seen in the previous section, the system is not provided with a usual fitness function: the values of chromosomes fitness are obtained by interacting with the agents and submitting them the current population of solutions.

A sketch of the system is shown in Figure 1 where the differences of our system with respect to a classical GA are in evidence.
3.1 Selection and Crossover

The evolution phase starts with the selection procedure, in which the chromosomes are selected for the crossover phase, using the standard roulette-wheel method, where the best individuals are more likely to be selected.

The crossover operator used in our algorithm is a one-point crossover. The gene sequence of the two chromosome parents are cut in two subsequences of $r$ and $m - r$ elements each, where $r$ is a random number between 1 and $m - 1$. Each of the two chromosome outbreeding is created by taking the union of two subsets coming from different parents. A repairing phase can be necessary after crossover in order to guarantee the unicity of components in the outbred products: if a son has one or more duplicated genes, the replacing components are taken from the parents.

3.2 Mutation

We define three different mutation operators. The first mutation operator behaves in an explorative way by replacing a component with a randomly chosen component which does not appear in any other chromosome. The second one replaces a component with a randomly chosen component among all the available components. The last one tries to exploit all the knowledge acquired in the evaluation phase by replacing a randomly chosen component preferring the components which are likely to give a higher contribution to the chromosome fitness. Since this information is not available to the system, a rough estimate of a quantity proportional to $d(c_i)$, for each component $c_i$, is computed by averaging the degree of all the chromosomes in which $c_i$ appears.

3.3 Static vs Dynamic

It is worth noticing that the algorithm has, in principle, no knowledge if it is trying to solve a static or a dynamic version of the problem. On the other hand there are some mutation strategies more suitable for static problems or for dynamic ones. For example, in the static case, it is possible, in some cases, to compute a more accurate estimate of the unknown fitness contribution $d(c_i)$ of a single component by solving a series of linear equations system. Each equation can be generated using the value $d(p_i)$ of each chromosome evaluation and the assumption that the genes contribution is additive. Unfortunately the method cannot be applied in general when the problem is dynamic or when the fitness contribution of genes is not additive.

Moreover the exploit mutation operator previously described does not work well in the dynamic version of the problem: the estimate could not be accurate because of possible changes in the components degree. Since the accuracy of this estimate depends on time, we are currently working on the idea of introducing a temporal factor in the estimation of the contribution of a single gene to the global chromosomes evaluation.

4 Experiments

In order to experiment the described algorithm for solving IP problems, a special module has been designed to simulate the evaluation given by the external agents and the dynamical changes of the domain during the time.

In the rest of this section we first describe how the simulation of a dynamic environment is performed, and then we will show and discuss the results we obtained.

4.1 Simulation Module

As we have previously said, the GA for IP problem submits to the evaluation module each individual of a given population. This module simulates the external fitness evaluation given by the clients/agents. According to the IP additive model, the evaluation of the individual is computed as the sum of the evaluation of the single genes. For instance, referring to the newspaper example, an individual is a single newspaper front page, and the evaluation is the sum of the values of its components, i.e. the values associates to the news in the front page. The initial evaluation assigns to each possible components a value following a normal distribution with mean 0.5 and variance 0.5/3 restricted to $[0, 1]$.

The dynamical changes are modelled by two parameters $(p_d, f)$, which respectively define the probability $p_d$ that a dynamical change happens in a given generation and the percentage $f$ of elements which are affected by the change in their values. At each generation, the evaluation module randomly decides with probability $p_d$ whether a change has to take place in the external fitness; if it is so, then the value of $\lfloor f \cdot N \rfloor$ components is randomly changed following the same normal distribution previously described.

It is worth noticing that changing the values of the elements in the gene domain can be interpreted as modeling different dynamical aspects of the domains:

- a change of agent preferences in a fixed domain of components, (e.g. newspaper readers changes their interests or the set of readers changes while the available news do not change)
- a dynamical change which affects the components (e.g. the newspaper readers do not change but the set of available news is changing).

The tests been conducted by fixing the genes domain, the number of components in a product and the population size parameters respectively to

$$N = 200, \ m = 10, \ n = 20$$

for T=1000 generations while changing the value of the other parameters. The experimentation should be considered preliminary, in order to better understand if and how the system performances are related to these parameters a more extensive experimentation is needed.
the most cases very good. It is important to recall that our objective is to maximize the total fitness during all the built generations, or in equivalent form, is to maximize the mean value of the fitness. In the best case shown in the graph the mean value obtained is 0.92, with respect to a maximum equal to 1; if we compute the mean value excluding the first 200 generations necessary to the algorithm as training phase, the value becomes 0.94.

4.3 Dynamic Simulation

In the second phase, the experimentation with DIP problem, we chose to run the system fixing the parameters $p_c = 0.15$ and $p_m = 0.09$ and varying dynamical change parameters $p_d$ and $f$. As said, the last two parameters simulate the change over the fitness values, so we called this phase dynamical simulation. These experiments were performed in a systematic way. First the probability of change in a given generation $p_d$ was varying and the percentage of change $f$ was fixed, then the former was kept fixed and the latter was varied.

The best performances were obtained when the amount of change $f$ is small. For the same value of $f$, better performances are reached by low values of $p_d$. These results meet our expectations: the smaller the change is, closer the results in dynamical case are to the static case. This behaviour has a very intuitive motivation. When some changes affect the fitness of some components, the algorithm loses information and the population grown in the immediately next generations can have smaller fitness values. The system has to adapt itself and has to try to recover the fall.

The results obtained from this simulations are very encouraging. For instance, the graph in Figure 3 shows the performances of the system when $f = 0.05$. The values of the parameter $p_d$ are shown in correspondence of the line depicting the mean fitness values we obtained. The line for $p_d = 0$ represents the static case, while the line for $p_d = 1$ represents the opposite case when the 5% of fitness is changed at each generation. It is worth to noticing that also in this case the algorithm works well. The mean value for the fitness is 0.67 with respect the initial mean value equal to 0.6, with an improvement of more than 10%.

A very interesting and significant result obtained in the experiments is that the overall fitness obtained by the algorithm, or equivalently the average of the population fitness, appears to be invariant with respect to the product $p_d \cdot f$, i.e. when the probability of a change in a single component is constant. The result has been obtained by systematically comparing the algorithm behaviour on a number of DIP problems while varying the parameters $f$ and $p_d$ and maintaining constant their product, an example table of the results is shown in the figure 1. Note that the expected number of 1500 changes in $T = 1000$ generations corresponds to $p_d \cdot T \cdot f \cdot N$.

A possible explanation of these phenomenon is that since $p_d$ is directly related with the time rate of dynamical

Figure 2. Results for the static production problem.

The experimentation was performed in two phases, a static simulation and a dynamic simulation.

4.2 Static Simulation

At first, a set of static IP problems (i.e. problems without variations in the fitness values) was run. In this case the dynamical change parameters $p_d$ and $f$ are both equal to 0, while crossover $p_c$ and mutation $p_m$ probabilities are varying. After some preliminary simulation we also decided to use only the explorative mutation operator.

This phase had two aims: to provide a preliminary estimation of the system behavior and to find the values of $p_c$ and $p_m$ guaranteeing the best performances.

Since the system results are highly depending on the randomization, we run each problem 10 times and then computed the average values. In the following we called problem an instance of the parametric problem defined in the Problem Description section. The results obtained from a problem instance are average values over 10 runs obtained by an instance of the problem.

Since the initial values of the fitness, although with a normal distribution, are assigned randomly the optimal solutions, slightly differs in different runs. In order to compare them we have normalized them with respect to 1 as maximum. A graph collecting some significative results and showing the system performances for the static problems is in Figure 2. Each line in the graph shows the mean values of the population fitness ($y$ axis) during the 1000 generations ($x$ axis). An interesting result of the SIP experimentation phase was that the system has better performances when the mutation probability $p_m$ assumes low values. Moreover the system behaviour is very similar in the runs with the same value for the $p_m$ parameter and varying $p_c$ values for crossover. In this cases, the smaller $p_c$ probability of crossover the higher the mean values of the fitness. However, there were some combinations of parameters that guarantee very high performance. In Figure 2 this combination is represented by the values $p_c = 0.15$ and $p_m = 0.09$. Since this result, we decided to use this setting for the dynamic simulation phase.

As the graph shows, the system performances are in
change, a low value of $p_d$ gives the algorithm more time to adapt to the changes. On the other hand, the decrease of the fitness is higher when a change happens since the amount of change is related to $f$. Conversely, with a high rate of small changes, the algorithm does not perform at its best, but the fitness decrement is smaller when changes happen.

This interpretation seems to be confirmed by another result shown in the figure 4, i.e. the variance of the average fitness increases when $p_d$ decreases.

5 Conclusions

An interactive and adaptive genetic algorithm have been presented, for a class of Dynamic Interactive Production problems characterised by component additive fitness. The goal of such algorithms is not to find a single optimal solution, but to have an average good behaviour in the continuous process of producing results in a changing environment. The experimental results are promising since the algorithm is able to adapt to the changes in a dynamical environment while maintaining an acceptable performance.

Experiments also reveal some properties which point out interesting features of the adaptive behaviour of the algorithm:

- the algorithm ability to recover to a given amount of changes seems to preserve the average fitness, despite of the distribution of the changes over the time
- frequent changes result in a lower variance in the algorithm performance

Further experimental investigations are needed to tune and refine the algorithm and to clarify the relationship among the problem parameters. Ongoing development regards the design of a mutation operator which better exploits the information of previous generations for estimating the fitness of single components.

Future work will regard different classes of Interactive Production problems which are worth to be investigated, such as problems in which components are categorized, i.e. the components are not interchangeable, but they are organized in functional categories which compose the product instance. Other extensions concern problems in which the external fitness is not additive with respect to the components, for instance by allowing different multiplicative factors for the components, and context dependent evaluations of components.

6 References