A Review of Evolutionary Algorithms for E-Commerce

Alex A. Freitas

PUC-PR
PPGIA-CCET
R. Imaculada Conceicao, 1155
Curitiba – PR. 80215-901. Brazil
alex@ppgia.pucpr.br
http://www.ppgia.pucpr.br/~alex

Abstract. Evolutionary Algorithms (EAs) are adaptive algorithms based on the Darwinian principle of natural selection. Intuitively, their adaptive nature makes them suitable for highly dynamic environments, which is often the case in e-commerce applications. This chapter presents a review of EAs for e-commerce. It starts by discussing the main characteristics of EAs in general. Then it discusses several EAs developed for e-commerce applications, focusing on three kinds of e-commerce related tasks, namely: information retrieval on the web, discovery of negotiation strategies and improvement of web-page presentation.

1 Introduction

E-commerce is a relatively new, interdisciplinary field, consisting of the integration of commercial activities with several areas of computer science, such as the world wide web, database systems, metadata and ontologies, agent-based systems, multimedia and visualization, information security and privacy, etc.. It should be noted that sometimes the terms e-commerce and e-business are used with different meanings, with the latter being used as a more general term, which includes the former. In this chapter we use the terms e-commerce and e-business interchangeably, following [25]'s chap. 16. To quote from the page 259 of that reference:

"As far as we are concerned, e-commerce can be considered to be broad such as putting up a web page or listening to music on the web or conducting transactions on the web."

This means that we are using the term e-commerce to refer not only to the process of carrying out commercial transactions on the web but also to activities that give support to this process, such as learning, training and entertainment on the web, displaying information on web pages in such a way that the web page is as nice as possible for the user, retrieving relevant information from the web, etc.

E-commerce is an interesting research topic, not only due to its strategic economic value [1], but also to its potential for research in advanced areas of computer science, such as data mining [14] and artificial intelligence (AI) in general.

In this chapter we explore the use of a paradigm of artificial intelligence (AI) techniques, called evolutionary algorithms (EAs), for e-commerce applications. The basic motivation, as usual in the case of an AI paradigm, is to increase the degree of computational "intelligence" of a system, making it more autonomous and more adaptive to changes in its environment.

The potential of e-commerce for AI research stems mainly from two facts. First, e-commerce systems are already automated. This means that many of the hurdles associated with system building (such as data collection, integration with other automated systems of the organization, etc.) are significantly lower in e-commerce, by comparison with the process of automating a manual commercial system from scratch. Second, e-commerce systems tend to be very dynamic. This is due to several reasons. For instance, the contents of the web changes very fast. In addition, in e-commerce customers who are currently visiting a given company's web site can switch to the web site of a competitor in a few seconds, which would be much more difficult in a physical commerce environment.

Hence, the application of EAs techniques in e-commerce seems a promising research direction. EAs are robust, adaptive techniques, which – at least in principle – have a good potential for coping with dynamic environments.

In this chapter we discuss evolutionary algorithms for three kinds of e-commerce tasks:

(a) information retrieval in the web;

(b) discovering negotiation strategies;

(c) improving the presentation of web pages.

Out of these three tasks, the second one – discovering negotiation strategies – seems the most related to the central activity of e-commerce, namely carrying out commercial transactions on the web. Indeed, people like to negotiate prices at e-stores [16]. The other two tasks are less related to this central goal, but they are important tasks to support the broader process of e-commerce. Indeed, an attempt to retrieve relevant information from the web (among the huge amount of information stored in the web), say relevant information about a given kind of product, is often one of the first steps performed by a customer interested in buying something on the web. In addition, the layout and other graphical aspects of a web page can have a significant impact on the decision of a customer to buy or not a product announced in a given web page.

This chapter is organized as follows. Section 2 presents a brief review of EAs. Section 3 discusses the application of EAs to the three above-mentioned tasks of e-commerce. Finally, section 4 presents a summary and some conclusions drawn from the discussion of section 3.

2 A Review of Evolutionary Algorithms (EAs)

Evolutionary Algorithms (EAs) is essentially the name given to a large class of computational problem-solving algorithms inspired by the principle of natural selection. The basic idea is that at each generation the fittest (the best) individuals of the current population survive and produce offspring resembling them, so that the population gradually contains fitter and fitter individuals.

This idea is used, at a high level of abstraction, as a basis for designing computational EAs, as follows. Each "individual" of an evolving population represents a candidate solution to a given problem. Each individual is evaluated by a fitness function, which measures the quality of its corresponding solution. Then these individuals evolve towards better and better individuals via operators based on natural selection, i.e. survival and reproduction of the fittest, and genetics, e.g. crossover and mutation operators.

There are several kinds of evolutionary algorithms (EA) proposed in the literature. For a comprehensive review of several kinds of EA the reader is referred to [3]. In this chapter we are mainly interested in genetic algorithms (GA) and genetic programming (GP), which seem to be the kinds of EA that have been the most used in e-commerce-related tasks. In particular, most EAs discussed in this chapter will be GAs, but two GPs will also be discussed here.

Hence, in the remainder of this section and in the next two subsections we present a brief introduction to GAs and GP, to make this chapter self-contained. As will be seen in the next subsections, GAs and GP differ mainly with respect to the representation of an individual, which in turn leads to some differences in genetic operators. However, both kinds of EA are still based on the same principle of natural selection and genetics. Indeed, at a high level of abstraction, both GAs and GP perform essentially the same sequence of steps, as shown in the generic pseudocode of Algorithm 1.

```
create initial population of individuals;
compute fitness of each individual;
REPEAT
   select individuals based on fitness;
   apply genetic operators to selected
     individuals, creating offspring;
   compute fitness of each offspring individual;
   update the current population;
UNTIL (stopping criterion)
```

Algorithm 1: Generic, abstract pseudocode for GA and GP

As shown in this pseudocode, the first step is to create a population of individuals. The initial population can be generated at random or by using some problem-dependent heuristic. Then the individuals of the initial population are

evaluated according to a fitness function, and the algorithm starts the REPEAT-UNTIL loop.

An important step of this loop is the selection of individuals based on fitness. In general the better the fitness of an individual (i.e., the better the quality of its candidate solution) the higher the probability of an individual being selected. There are several different selection methods that can be used to implement this basic idea. Here we mention just one, tournament selection [5], which is both simple and effective. For a more comprehensive discussion of selection methods the reader is referred to [3]. In tournament selection the EA randomly chooses k individuals from the current population, where k is a parameter called the tournament size. Then the k individuals "play a tournament" to decide which of them will be selected to produce offspring. The winner of the tournament can be chosen either in a deterministic manner or in a probabilistic manner. In the former case the winner is simply the individual with the best fitness among the k individuals playing the tournament, whereas in the latter case each of the k individuals can be chosen as the winner with a probability proportional to its fitness. Deterministic tournament seems more common in practice.

Once individuals are selected, the next step of Algorithm 1 is to apply genetic operators to the selected individuals (parents), in order to produce new individuals (offspring) that, hopefully, will inherit good genetic material from their parents. This step will be discussed in sections 2.1 and 2.2 separately for GA and GP, since these two kinds of EA usually require different genetic operators – due to their different individual representations.

Then the fitness of each of the new individuals is computed, and another iteration of the REPEAT-UNTIL loop is started. This process is repeated until a given stopping criterion is satisfied. Typical stopping criteria are a fixed number of iterations (generations) or the generation of an individual representing a very good solution.

It is worthwhile to mention that EAs perform a *global* search in the space of candidate solutions, by contrast with the local search performed by hill climbing-like, greedy search algorithms [11]. In particular, EAs work with a population of candidate solutions, rather than working with a single candidate solution at a time. This, together with the fact they they use stochastic operators to perform their search (as will be seen below), tend to reduce the probability that they will get stuck in local maxima, and increase the probability that they will find the global maximum in the space of candidate solutions.

2.1 Genetic Algorithms

Recall that an individual corresponds to a candidate solution to a given problem. In GAs an individual is usually a linear string of "symbols", often called "genes". A gene can be any kind of symbol, depending on the kind of candidate solution being represented. For instance, in GAs for information retrieval (to be discussed in section 3.1) a gene can be a word found in a text, or a pair composed by a word and its importance weight.

In general the main genetic operator of GAs is the crossover operator. It essentially consists of swapping genes between (usually two) individuals [13], [18]. Figure 1 illustrates a simple kind of crossover, called one-point crossover. Figure 1(a) shows two individuals, called the parents, before crossover. A crossover point is randomly chosen, represented in the figure by the symbol "|" between the second and third genes. Then the genes to the right of the crossover point are swapped between the two parents, yielding the new individuals shown in Figure 1(b).

X1 X2 | X3 X4 X5	X1 X2 | Y3 Y4 Y5
Y1 Y2 | Y3 Y4 Y5	Y1 Y2 | X3 X4 X5
(a) Before crossover	(b) After crossover

Figure 1: Example of one-point crossover

X1 | X2 X3 | X4 X5	X1 | Y2 Y3 | X4 X5
Y1 | Y2 Y3 | Y4 Y5	Y1 | X2 X3 | Y4 Y5
(a) Before crossover	(b) After crossover

Figure 2: Example of two-point crossover

Figure 2 illustrates another kind of crossover, called two-point crossover. Now a pair of crossover points is randomly selected in each of two parent individuals. In each parent the genes lying between the pair of crossover points are considered a block to be swapped, as a whole, by the crossover operator.

In addition to crossover, which is the main genetic operator in GAs, it is also common to use mutation. In essence mutation replaces the value of a gene with a new randomly-generated value (among the values that are valid for the gene in question). Note that mutation can yield gene values that are not present in the current population, unlike crossover, which swaps existing gene values between individuals.

Both crossover and mutation are stochastic operators, applied with user-defined probabilities. In GAs the probability of mutation is usually much lower than that of crossover, in part due to an analogy with natural selection and genetics, where mutations are rare and usually harmful for the organism.

2.2 Genetic Programming

As mentioned above, the main difference between GAs and GP is in the individual representation used by each kind of EA, which in turn leads to some differences in the genetic operators used by each of them.

First of all, most GP algorithms use a tree-based individual representation, and this will be the only kind of GP representation discussed in this chapter. For a discussion of other kinds of GP representation the reader is referred to [4]. Each individual is represented by a tree consisting of two kinds of nodes: internal nodes, containing functions, and terminal (leaf) nodes, containing variables of the problem being solved or constants.

It should be noted that an individual's tree can vary a lot in both size and shape, unlike the linear strings of "conventional" GAs, which usually have a fixed length (although there are several exceptions, where the length of the string is variable). Most important, in GP an individual (candidate solution) consists not only of problem-specific variables and/or their values (data) but also functions (operators applied to the variables/values) – unlike GAs, where an individual's candidate solution usually consists only of variables/values. Hence, a GP individual is often called a "program", even though in many cases this term should be interpreted in a loose sense. In any case, the distinction between GA and GP is blurring as there is a growing tendency towards the unification of the field of EAs [9].

The set of all functions (or operators) available to be used in internal tree nodes is called the function set, whereas the set of all variables and/or variable values available to be used in leaf nodes is called the terminal set.

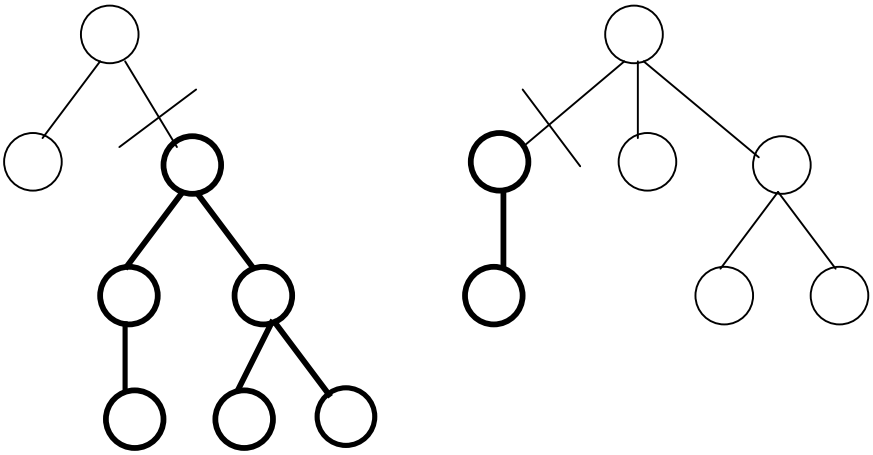
In general the function set of a GP algorithm must satisfy at least two properties, namely sufficiency and closure [15]. Sufficiency means that the function set's expressive power is good enough to be able to represent a correct solution (or at least a very good solution) to the target problem. Closure means that each function of the function set should be able to accept, as input, any output produced by any function in the function set.

GP crossover essentially swaps two subtrees between two parent individuals. In each parent the subtree to be swapped is usually chosen at random. This kind of subtree crossover is illustrated in Figure 3, where the "crossover point" is indicated by a tilted line and the subtrees swapped by crossover are shown in bold.

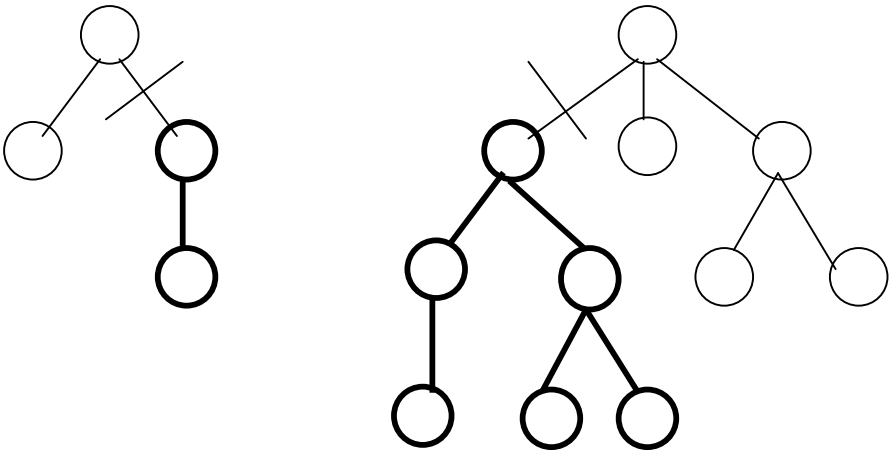
In addition to crossover, mutation is also often used in GP. There are several kinds of GP mutation operator. We briefly mention here only two kinds, namely point mutation and subtree mutation. A more comprehensive discussion on several kinds of GP mutation operators can be found in [4] (pp. 240-243), [6].

Point mutation is the simplest form of mutation. It replaces a single node in a tree with another randomly-generated node of the same kind. By "same kind" we mean that an internal node is replaced by another internal node (with another function) and a terminal node is replaced by another terminal node (with another variable or value). Subtree mutation randomly selects an internal node in the tree, and then it replaces the subtree rooted at that node with a new randomly-generated subtree. The new randomly-generated subtree should be created subject

to some restrictions of depth and/or size, to avoid the generation of a too large subtree.



(a) Two parents before crossover



(b) Two children (offspring) produced by crossover

Figure 3: Conventional tree crossover in genetic programming

2.3 Co-Evolution

Co-evolution means the complementary evolution of two or more populations [21]. In nature a typical example is the co-evolution of predators and preys, where new features developed by predators trigger an evolutionary response in the preys.

In the context of EAs, the basic idea is that the individuals of each population evolve to be adapted to the individuals of the other population(s). Co-evolution involves a dynamic fitness function, where the fitness of an individual in a given population is determined by how adapted that individual is to individuals in other population(s). Hence, the same individual can have a high value of fitness in one generation but a low value of fitness in another generation, depending on the current contents of the other population(s).

This basic concept of co-evolution will be particularly useful to understand the application of EAs to the problem of evolving negotiation strategies in an e-commerce framework, as will be seen in section 3.2.

2.4 Interactive Evolutionary Algorithms

In some problems the quality of the candidate solution represented by an individual is inherently subjective, and it would be very difficult to design a good fitness function specified by a well-defined, objective mathematical formula. In such cases it is natural to use an interactive fitness function. The basic idea is that the fitness of an individual is directly evaluated by a human user, based on his/her own subjective preferences.

Two applications of interactive fitness function are briefly mentioned here. One involves an image-enhancement application [22], where the user drives GP by deciding which individual should be the winner in tournament selection. The other application involves attribute selection for the well-known classification task of data mining. In this application each individual of a GA represents a candidate attribute subset [24]. In order to evaluate an individual its candidate attribute set is given to an algorithm that discover classification rules, and the quality (fitness) of these rules are interactively and subjectively evaluated by the user.

In the context of e-commerce an interactive fitness function can be naturally used, for instance, to allow a user to subjectively evaluate how “nice” a given web page is. An example of this application will be discussed in section 3.3. Another example of the use of interactive fitness function will be discussed in section 3.1, in information retrieval.

3 Applying Evolutionary Algorithms (EAs) to E-Commerce

This section is divided into three subsections, according to the kind of e-commerce-related problem for which the EA was designed. Subsection 3.1 discusses EAs for information retrieval in the web. Subsection 3.2 discusses EAs for negotiation strategies. Subsection 3.3 discusses EAs for improving the presentation of web pages.

3.1 Information Retrieval in the Web

In this section we discuss evolutionary algorithms (EAs) designed for retrieving information in the web. In general the EAs discussed in this section are genetic algorithms (GAs).

Before we proceed, we present a brief introduction to information retrieval. The basic idea is as follows. Given a query, representing the interest of the user, and a set of documents, the system must select the documents most relevant to the user's query. The query can take different forms, from a small set of keywords to an entire text. In any case the contents of the query is compared with the contents of each document in the document base, and a similarity measure is computed for each document. The documents most similar (most relevant) to the query are returned to the user, in decreasing order of similarity. In the context of this section, each web page can be considered a document.

One important point in information retrieval is how to represent the contents of a document. A popular form of representation of documents is the vectorial model [23]. In this model each document is represented by a multi-dimensional vector, where each dimension corresponds to a term (usually a single word). For each document, the value of each dimension of its vector typically depends on the frequency of occurrence of the corresponding term in the document (the term frequency) or some related measure. Hence, the similarity between the query and each document can be naturally measured by the angle formed by the respective two vectors – i.e., the smaller the angle formed by a query vector and a document vector, the more similar the query and the document are.

3.1.1 GAs Without Focus on Web Page Structure

In this subsection we discuss two GAs for information retrieval in the web with two different individual representations. However, both GAs have in common the fact that they do *not* focus on the use of information about web-page structure. By contrast, in subsection 3.1.2 we will discuss a GA that focus on the use of information about web-page structure.

In [2] an individual represents the profile of a user's interest, consisting of a set of terms and their corresponding weights. Hence, the GA tries to find the best possible combination of terms (words) and term weights that allow the retrieval of documents relevant for the user.

More precisely, an individual consists of a variable-length list of genes. Each gene is a term-weight pair, i.e. a term and its associated weight. This structure is illustrated in Figure 4. This representation also includes an intuitive semantic-integrity constraint, namely the fact that a given term can occur at most once in the genome of a given individual. We will call this a term-uniqueness constraint.

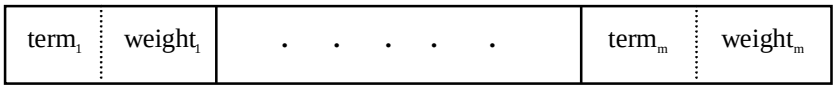


Figure 4: Genome consisting of term-weight pairs for information retrieval

In GAs constraints can be handled in at least three ways: (a) penalizing the fitness of "invalid" individuals, i.e., individuals that violate the constraint; (b) using special repair operators that transform an invalid individual into a valid one; or (c) avoiding the creation of an invalid individual altogether, by not applying a genetic operator when it would produce an invalid individual. The third approach (c) seems to be the one used in [2]. Crossover and mutation are performed only if they produce valid offspring, respecting the term-uniqueness constraint.

The crossover operator is based on the idea of two-point crossover, as discussed in section 2.1.

There are two kinds of mutation operator, weight mutation and term mutation. Weight mutation randomly modifies the value of the weight in a gene. Term mutation randomly modifies both the term and its weight value in a gene.

We now turn to the work of [20]. In this work an individual also represents a kind of profile of a user's interest, like in [2]. In both systems an individual essentially represents a query, but there are two important differences between these systems, as follows.

1) In the former an individual consists of only a set of terms of a query, whereas in the latter an individual consists of a set of terms and their corresponding weights, as discussed above.

2) The GA proposed by Morgan & Kilgour was designed to support on-line information retrieval in a dynamic environment where user interests are continuously changing, whereas the GA proposed by Atsumi seems to cope with a somewhat less dynamic environment.

Let us discuss each of these points in turn. First, Morgan & Kilgour's GA use an individual representation consisting of a set of query terms, such as {"philosophy", "science", "induction"}. Note that this representation does not include term weights. Each individual is represented by a fixed-length list of terms (genes), but some genes can contain a special symbol denoting an "empty" gene, with no associated term. Therefore, each individual effectively represents a variable number of terms. For instance, the above-mentioned individual, containing a set of three terms, might be internally represented by a 5-gene genome such as {" philosophy", "science", "induction", [], []}, where the symbol

"[]" denotes an empty gene. (The number of genes per individual suggested by the authors is 10.)

Second, in Morgan & Kilgour's GA the fitness of an individual (query) is evaluated by the user's subjective judgement, which is based on the usefulness of the information retrieved by the query. Note that this is quite different from the more objective rates of recall and precision used by most information retrieval systems, including the above-described system proposed by [2]. To reduce the amount of time that the user spends evaluating query results the system works with a small population of 10 individuals.

The system is an interactive GA (section 2.4) that continuously interacts with the user, as follows. Since the GA is run on-line, the user can submit new queries to the database of texts during the GA run. Whenever the user submits a new query, that query is added to the current population of system-created queries maintained by the GA. The new individual (query) is assigned a maximum value of fitness, based on the assumption that user-created queries are the most important kind of information for building a profile of a user's interest.

Both user-created queries and system-created queries are used to produce the next population of (system-created) queries via the conventional process of selection, crossover and mutation. An one-point crossover operator (section 2.1) recombines the terms of two parents to produce two offspring.

There are two kinds of mutation, namely random mutation and synonym mutation. The former replaces terms by randomly-chosen words from the UNIX spell dictionary. The latter replaces terms by one of their synonyms from the WordNet synonym dictionary [10]. The system uses high rates of mutation – on the order of 50%. Hence, in this system mutation is a major operator to explore the search space – unlike conventional GAs, where mutation is usually a minor search operator.

3.1.2 A GA with Focus on Web Page Structure

In the previous subsection we have discussed two projects which proposed GAs for information retrieval in the web. However, in both projects the use of information about web page structure was limited. We now discuss a GA for information retrieval on the web with considerable more focus on the use of web page structure [7].

The basic idea is to take into account not only the frequency of terms in web pages, but also information about the relative importance of each term as indicated by the web page structure. More precisely, for each term the GA takes into count the kind of HTML tag in which that term occurs. Cutler et al. grouped HTML tags into six classes, namely: Plain Text, Strong, List, Header, Anchor and Title. Intuitively, the relative importance of a term, for information retrieval purposes, depends on the class of tag in which that term occurs. For instance, a term occurring in a Title tag tends to be more important than a term occurring in a Plain Text tag. The question is how to quantify the importance of each tag class. This is essentially the task addressed by the GA.

Each individual of the GA represents a vector of six weights, called a class importance vector (CIV). Each position i (i=1,...,6) of this vector, corresponding

to a gene, contains the importance weight assigned to the *i*-th tag class. Hence, the GA tries to find the best possible combination of tag-class importance weights.

In order to evaluate the fitness of an individual, the information in its CIV has to be combined with information about term frequencies in web pages. This is done as follows. For each term and for each web page in which that term occurs the system computes a term frequency vector (TFV). Each position *i* (*i*=1,...,6) of this vector contains the frequency of occurrence of the term in the *i*-th tag class. When an information retrieval query is submitted, the system computes the weight of terms in web pages according to the formulas:

$$w = tf \cdot idf,$$

where *idf* is the well-known inverse document frequency [23] and *tf* (term frequency), which represents the importance of a given term to a given document, is given by *tf* = TFV **.** CIV, where "**.**" denotes the inner product of the two vectors TFV and CIV.

Hence, the better the CIV represented by an individual, the better the results of the information retrieval query will be. The fitness of an individual is computed by an information retrieval metric based on recall and precision. Hence, the fitness measure is objective, like in [2] but unlike in [20].

Computational experiments showed that the terms in the Strong and Anchor classes had the highest weights in the best CIV evolved by the GA. Hence, these two classes of tags were found to be the most useful ones for improving the effectiveness of information retrieval in the performed experiments.

3.2 Co-Evolving Negotiation Strategies

In this section we discuss two evolutionary algorithms for evolving negotiation strategies. Both algorithms are based on the use of co-evolution, as discussed in section 2.3.

[8] studied the co-evolution of agents representing bargaining strategies. Their study involved a simple, artificial game problem consisting of three agents which competitively negotiate to form a two-agent coalition. The agents bargain with each other to decide which two-agent coalition will be formed and how the points constituting the coalition's value will be divided between the two agents forming the coalition. The agent who is excluded from the coalition receives zero.

For instance, in a given game the value *v* of each of the three possible coalitions are as follows: $v(AB) = 18$, $v(AC) = 24$, $v(BC) = 30$, where A, B, C denote the three agents and $v(X,Y)$ denotes the value of the coalition formed by agents $X,Y \in \{A, B, C\}$. If, say, A proposes to form a coalition with B, A makes an offer to B. Such an offer specifies how the 18 points of the coalition are to be divided between A and B.

Hence, a game proceeds as follows. An agent (the initiator) makes an offer to another agent (the responder). If the responder accepts the offer the game ends, and each of those two agents receives a part of the coalition points, as specified in the offer, while the third agent receives zero. If the responder does not accept the offer then it becomes the initiator and makes an offer to either of the two other

agents. The game proceeds in this fashion until an offer is accepted or a maximum number of offers has been rejected – in which case all three agents receive zero.

Each agent consists of a set of alternative strategies for playing the above-described game. Each agent's strategies are represented by a population of 50 genetic programming (GP) individuals. In essence, each individual (strategy) is an IF-THEN-ELSE statement of the form:

IF {*condition*} THEN {*action-1*} ELSE {*action-2*}.

The *condition* part of the strategy is a triple of the form {*Player*, *LB*, *UP*}, which evaluates to true if, in the current received offer, the amount of points assigned to *Player* is between the values *LB (Lower Bound)* and *(Upper Bound)*, inclusive. Otherwise the condition evaluates to false. *Action-1* and *action-2* can be one of the following three statements: (a) the *ACCEPT* symbol, indicating that the offer is accepted and the game ends; (b) an offer; or (c) another IF-THEN-ELSE statement – so that complex strategies can be specified by nesting IF-THEN-ELSE statements. An offer is a pair of the form {*Player*, *V*}, meaning that the current agent offers to *Player* (another agent) *V* points.

For instance, suppose that agent B is responding to an offer with the strategy IF {B 4 9} THEN {ACCEPT} ELSE {A 10}. If the amount of points assigned to B in the received offer is between 4 and 9 (inclusive) then B accepts the offer. Otherwise it offers A 10 points.

Recall that an agent (player) is represent by a population of 50 individuals (strategies). In order to compute the fitness of an individual, its strategy plays against a number of combinations of strategies of the two other agents (players). In other words, the fitness of a given individual in a given population is determined by how adapted that individual is to the individuals in the other two populations (representing other players' strategies), characterizing a co-evolution scheme – see section 2.3.

It should be noted that, although the above-described coalition game simulates an environment with competing bargaining strategies found in e-commerce applications, the simulation is still oversimplified, for several reasons, including the following ones.

First, the number of points assigned to an agent is a single scalar value. In reality agents are likely to be interested in maximizing the value of several different, possibly-conflicting aspects (or issues) of a commercial agreement. Such aspects may include the price of a product, its quality, time to delivery, etc. Second, it ignores important factors of real-world commercial agreements, such as the length of time available for an agent to reach an agreement.

We now discuss a project where the problem being solved is a less simplified, more realistic model of e-commerce applications. In this project [17] each agent has a negotiation strategy that takes into account the values of several different issues. More precisely, each agent a has a scoring function $V_{a,j}$ for each issue j. This function determines the score (in the interval [0,1]) that agent a assigns to the value x of issue j specified in a given offer. An offer consists essentially of a vector of values, one value for each issue. When an agent receives an offer, it rates it by using a utility function $V_a(\boldsymbol{x})$ that is a linear combination of the values of the issues specified in the offer, i.e.:

$$V_a(\boldsymbol{x}) = \sum_j w_{aj} V_{aj}(x_j),$$

where w_{aj} is the importance (weight) of issue j for agent a. If the value of this utility function for the received offer is greater than the corresponding value of the counter offer that the agent would send at this point, then the offer is accepted. Otherwise a counter offer is submitted – unless the deadline assigned for negotiation has been reached.

The question is how to generate an offer or a counter offer. In essence, (counter) offers are generated by linear combinations of functions called tactics. Each tactic generates a value for a single negotiation issue based on a single criterion. There are 3 basic criteria used by tactics, namely the time remaining for reaching an agreement, the amount of resources remaining and the behavior of the opponent (which is determined by its tactics). In general as the deadline for reaching an agreement approaches and the quantity of resources is reduced an agent becomes progressively more likely to concede. Behavior-dependent tactics are variations of tic-for-tat that differ with respect to which aspect of the opponent's behavior they imitate and to what extent.

Each tactic is assigned an importance weight. For each negotiation issue, its value in an offer is specified by a linear, weighted combination of tactics. In other words, an agent can use a different weighted combination of tactics for each negotiation issue. Finally, each agent is associated with a strategy which, over time, varies the weights of the different tactics for each of the negotiation issues in order to adapt to changing environmental conditions – such as the observed behavior of the opponent.

This is where a GA comes in. The GA evolves a population of individuals, where each individual represents an agent. The genetic material of an individual represents the parameters of the negotiation tactics and their importance weights.

The fitness of an individual (agent) is determined by how well that individual performs with respect to other individuals. More precisely, there are two kinds of agents, buyers and sellers. To compute an agent's fitness the system performs a round-robin tournament where each buyer negotiates with each seller, following the basic idea of co-evolution discussed in section 2.3.

3.3 EAs for Improving the Presentation of Web Pages

In this section we discuss two GAs developed for improving the presentation of web pages. The first one is an interactive GA for generating HTML style sheets that are as nice as possible from the (subjective) viewpoint of a user, whereas the second one is a GA for improving the layout of a web page in more objective terms. Both systems assume that the contents of a web page has already been determined. What the systems try to find is the best visual presentation of that contents.

Let us start with the GA for generating HTML style sheets proposed by [19]. Each individual of the GA population represents an HTML style sheet. The genetic material of an individual consists of 26 characteristics determining the

look of a web site. Each characteristic is represented by a gene. These characteristics can be divided into two broad groups, namely:

(a) 5 global characteristics of an HTML page: the background, color of links, rules, bullets and arrows ("back", "next", "home"); and

(b) 21 characteristics determining the appearance of the text and paragraphs – these include, for instance, the font and color of text in title and paragraph tags.

The main question is how to compute the fitness of an individual – i.e., how to evaluate the "quality" of a style sheet. The author's proposed solution for this problem is to use an interactive GA. The motivation for this is that the quality of a style sheet strongly depends on subjective preferences of the user, which could hardly be mathematically defined. Hence, an individual's fitness is computed by presenting its style sheet to the user in a graphical way and letting the user explicitly select the style sheets that (s)he favors.

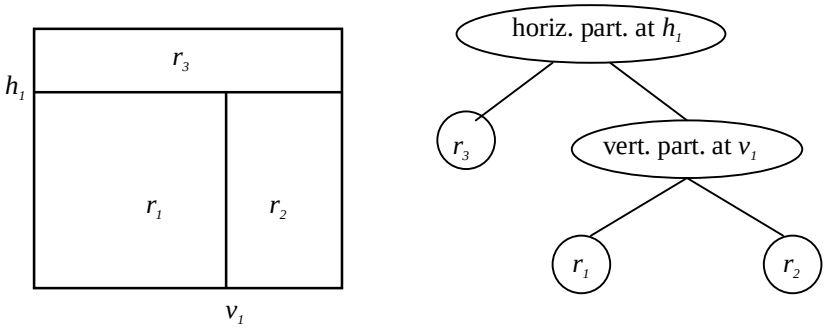
In order to facilitate the comparison of individuals (style sheets) by the user the system displays the whole population of individuals simultaneously. Of course, this severely limits the number of individuals in the population. In order to solve this problem the authors use a non-standard GA which uses a vector of probabilities to model an infinite population.

The basic idea is that the GA explicitly manages the probability of occurrence of each gene value. Let L be the number of genes of an individual. As discussed above, $L = 26$. Let k_i, i=1,...L, be the number of possible values of the i-th gene, and let $p(V_{ij})$ be the probability that the i-th gene has its j-th value, i=1,...L and j=1,...,k_i. The GA initializes the probability values so that, for each gene, all its values are equally likely to occur. Mathematically, $p(V_{ij}) = 1 / k_i$, $\forall i,j$. Then 12 individuals are generated, and their style sheets are applied to the web pages provided by the user, producing 12 pages that are simultaneously shown to the user. Next, the user selects the individuals corresponding to the style sheets that (s)he likes the most. (The user may also directly modify the genes of an individual in order to improve its style sheet. Such modifications are considered mutations.) The set of individuals selected by the user, which are supposed to contain the best gene values (characteristics of style sheets), are then used to update the probability values $p(V_{ij})$. This is done in such a way that the probability values $p(V_{ij})$ for the gene values occurring in the selected individuals is increased. Once the new values of $p(V_{ij})$ have been computed the next generation of individuals is generated. This new generation includes all individuals selected by the user in the previous generation, without modification, and the remaining individuals are generated based on the updated probability values $p(V_{ij})$. This iterative process is repeated until the user is satisfied with the style sheets proposed by the GA.

Let us now discuss the GP to automate the layout of web pages proposed by [12]. The system was designed to optimize the layout of web pages consisting mainly of pictures in the form of rectangles. The problem is to arrange $n \geq 1$ rectangles (pictures) $r_1,...,r_n$ on a larger rectangle (web page) R in such a way that some layout-quality criterion (discussed later) is optimized.

A layout for the web page is obtained by subdividing it into n rectangles using a recursive binary-partition method. Hence, a layout is represented by a binary tree with n leaves, each of them representing a rectangle r_i. A simple example is

shown in Figure 5, involving three rectangles r_1, r_2, r_3. Figure 5(a) shows a possible layout arranging those three rectangles on the larger rectangle, and Figure 5(b) shows a high-level view of the GP individual – in the form of a binary tree – representing the solution shown in Figure 5(a). The root node indicates that the rectangle R is horizontally partitioned, at coordinate h_1, into two subrectangles. The first of these subrectangles is simply r_3, as indicated by the root's left child (a leaf node). The root's right child is an internal node, indicating that the second subrectangle is vertically partitioned, at coordinate v_1, into two subrectangles. These subrectangles are simply r_1 and r_2, since both children of this internal node are leaf nodes.



(a) A candidate layout (b) GP individual representing layout in (a)

Figure 5: Arranging three rectangles on a larger rectangle via GP

In order to put a rectangle r_i into a given subrectangle produced by a GP individual it might be necessary to scale down r_i - i.e. to reduce its width and length in order for it to fit into the corresponding subrectangle. As pointed out by Fuchs, in the context of web pages such scaling down is possible and sensible, as long as the amount of scaling down is not excessive.

The question is how to determine the fitness function for this layout-optimization problem. The basic idea is that the fitness function must take into account two kinds of factor, namely the amount of left-over blank space in the layout (i.e. space in the web page that is not used by any picture) and the amount of scaling down used to fit the pictures into the web page. Both factors are to be minimized – i.e. the smaller their values, the better the corresponding candidate layout.

4 Summary and Discussion

In this section we summarize the main characteristics of the EAs reviewed in the previous section and provide a comparative analysis of those algorithms.

Table 1 summarizes the main characteristics of the above-discussed EAs. This table contains seven rows, one for each of those algorithms. The table contains five columns. The first column simply mentions the bibliographical reference for the corresponding EA. The algorithms are listed in the table in the same order as they were discussed in section 3.

Table 1: Summary of EAs for E-Commerce-related tasks

Reference	**EA**	**Task**	**Individual representation**	**Fitness**
[2]	GA	information retrieval	a set of term-weight pairs	objective, static
[20]	GA	information retrieval	a set of terms	subjective, interactive, dynamic
[7]	GA	information retrieval	a set of tag-class weights	objective, static
[8]	GP	negotiation	a negotiation strategy	competitive, dynamic
[17]	GA	negotiation	parameters of negotiation tactics	competitive, dynamic
[19]	GA*	web page presentation	an HTML style sheet	subjective, interactive, dynamic
[12]	GP	web page presentation	the layout of a web page	objective, static

* non-standard GA, using a vector of probabilities to model a population

The second column specifies the kind of EA, either a GA or GP. As can be observed in the table, most systems are GAs, but there are also two GP systems. We emphasize that the categorizations used in the cells of Table 1 are not absolute. They serve as rough approximations of their corresponding characteristics, for pedagogical purposes. For instance, as mentioned above, the distinction between GA and GP is blurring. In addition, the GA used in [19] is a non-standard, different kind of GA, since it uses a vector of probabilities to model a population. In any case, we believe the contents of Table 1 – particularly the fourth and fifth columns – are useful to draw some conclusions about trends and important issues in the application of EAs to the e-commerce-related tasks studied in this chapter.

The third column of Table 1 specifies the task being performed by the corresponding EA, which is either information retrieval in the web (subsection

3.1), or evolving negotiation strategies (subsection 3.2), or improving web page presentation (subsection 3.3).

The fourth column indicates the individual representation used by the corresponding EA. Of course, the individual representation depends on the kind of task being performed, and even within the same kind of task there are significant differences between the representations of different algorithms. Two interesting remarks can be made about the contents of this column. First, out of the three GAs for information retrieval in the web mentioned in the table, only the one proposed by [7]) focus on the use of information about web-page structure. More precisely, this system identifies six classes of tags found in web pages: Plain Text, Strong, List, Header, Anchor and Title. The system takes into account the class of tag in which each term (word) occurs, and explicitly searches for an optimum vector of importance weights for the tag classes. Second, the two EAs for evolving negotiation strategies use quite different individual representations, although their target task is quite similar. One of these EAs is a GP system, where a negotiation strategy is a (possibly nested, complex) list of IF-THEN-ELSE statements; whereas the other EA is a relatively simple GA, where only parameters of negotiation tactics (rather than the structure of a negotiation strategy itself) are evolved. On the other hand, the problem being solved by the GA seems to a more "realistic" model (although still just a model) of an e-commerce environment. The problem being solved by the GP seems to be "oversimplified", ignoring several important issues of a real-world e-commerce environment. In the future it would be interesting to see the use of GP to search for negotiation strategies in a more realistic e-commerce model.

Finally, the fifth column of Table 1 indicates the kind of fitness function used by the corresponding EA. There are three possible values for the cells of this column: an objective, static fitness function; a subjective, interactive, dynamic one; or a competitive, dynamic one. It is interesting to note that, out of the seven EAs, only three use an objective, static fitness function. The term "static" is used here to indicate that the fitness of an individual has the same value regardless of the time (generation number) in which it is computed and regardless of the other individuals in the population(s). By contrast, the value of a dynamic fitness function for an individual can change depending on the time in which it is computed and/or on the other individuals in the population(s). Four out of the seven EAs – i.e. the majority of them – use some kind of dynamic fitness function, more precisely either a competitive fitness function (in a co-evolution framework) or a subjective, interactive fitness function (whose value is explicitly determined by the user). Note that in the co-evolution frameworks studied here the fitness function is naturally dynamic, since the fitness of an individual (a negotiation strategy or parameters of a negotiation tactic) is determined by how adapted that individual is to individuals in the other population. Similarly, a subjective, interactive fitness function is naturally a dynamic one, since a user's interests and evaluations are continuously changing – and can even be inconsistent from one generation to another or among a group of individuals of a given generation.

To summarize this important conclusion: dynamic fitness functions – either competitive ones in a co-evolution framework or interactive ones – tend to have an important role in several e-commerce applications, due to the inherently

dynamic nature of this kind of application. In particular, co-evolution seems naturally useful for evolving negotiation strategies, and subjective, interactive fitness functions are intuitively useful in tasks such as improving web-page presentation.

As a final remark, the potential of EA for e-commerce applications goes considerably beyond what has been discussed in this chapter. The area of "evolutionary e-commerce systems" is still in its infancy, and it is likely that significant progress will be made in the next few years, motivated by at least two factors: (a) a growing interest in both e-commerce and evolutionary algorithms – from both academia and industry; (b) the need for robust, adaptive AI techniques (such as EA) that can significantly increase the degree of automation and computational "intelligence" of e-commerce systems, making them better prepared for effectively coping with the dynamic environments typically found in e-commerce applications.

References

1. Akkermans H. (2001) Intelligent E-Business: from technology to value. *Guest Editor's Introduction. IEEE Intelligent Systems, July/August 2001*, pp 8-10.
2. Atsumi M. (1997) Extraction of user's interests from web pages based on genetic algorithm. *English version of the original Japanese paper published in IPSJ SIG Notes (Information Processing Society of Japan, The Special Interest Groups Notes),* 97(51), pp 13-18.
3. Back T, Fogel DB and Michalewicz T. (Eds) (2000) *Evolutionary Computation 1: Basic Algorithms and Operators*. Institute of Physics Publishing, Bristol.
4. Banzhaf W, Nordin P, Keller RE, and Francone FD. (1998) *Genetic Programming ~ an Introduction: On the Automatic Evolution of Computer Programs and Its Applications*. Morgan Kaufmann, San Mateo, CA.
5. Blickle T. Tournament selection. (2000) In: Back T, Fogel DB and Michalewicz T. (Eds) *Evolutionary Computation 1: Basic Algorithms and Operators*, pp 181-186. Institute of Physics Publishing, Bristol.
6. Cavaretta MJ and Chellapilla K. (1999) Data mining using genetic programming: the implications of parsimony on generalization error. *Proc. 1999 Congress on Evolutionary Computation (CEC-99)*, pp 1330-1337. IEEE Press, Piscataway, NJ.
7. Cutler M, Deng H, Maniccam SS and Meng W. (1999) A new study on using HTML structures to improve retrieval. *Proc. 11th IEEE Int. Conf. on Tools with Artificial Intelligence*, pp 406-409. IEEE Press, Piscataway, NJ.
8. Dworman G, Kimbrough SO and Laing JD. (1996) Bargaining by artificial agents in two coalition games: a study in genetic programming for electronic commerce. *Genetic Programming 1996: Proc. 1st Annual Conf. (GP-96)*. Morgan Kaufmann, San Mateo, CA.
9. De Jong K. (2000) Evolutionary computation: an unified overview. *2000 Genetic and Evolutionary Computation Conf. Tutorial Program*, pp 471-479. Las Vegas, NV.

10. Fellbaum C. (Ed) (2000) *WordNet: an Electronic Lexical Database*. MIT Press, Cambridge, MA.
11. Freitas AA. (2001) Understanding the crucial role of attribute interaction in data mining. *Artificial Intelligence Review*, 16(3), pp 177-199, Nov 2001.
12. Fuchs MM. (2000) An evolutionary approach to support web page design. *Proc. 2000 Congress on Evolutionary Computation (CEC-2000)*, pp 1312-1319. IEEE Press, Piscataway, NJ.
13. Goldberg DE. (1989) *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison-Wesley, Reading, MA.
14. Kohavi R. and Provost F. (Eds) (2001) Special Issue on Applications of data mining to electronic commerce. *Data Mining and Knowledge Discovery 5(1/2)*, pp 5-10. 20. Jan/Apr 2001.
15. Koza JR. (1992) *Genetic Programming: on the programming of computers by means of natural selection.* MIT Press, Cambridge, MA.
16. Lina F-R and Chang K-Y. (2001) A multiagent framework for automated online bargaining. *IEEE Intelligent Systems*, pp 41-47, *July/August 2001*.
17. Matos N, Sierra C, Jennings NR. (1998) Determining successful negotiation strategies: an evolutionary approach. *Proc. 1998 Int. Conf. on Multi-Agent Systems (ICMAS-98)*, pp 182-189.
18. Michalewicz Z. (1996) *Genetic Algorithms + Data Structures = Evolution Programs*. 3rd Ed. Springer-Verlag, Berlin.
19. Monmarche N, Nocent G, Slimane M, Venturini G and Santini P. (1999) Imagine: a tool for generating HTML style sheets with an interactive genetic algorithm based on genes frequencies. *Proc. 1999 IEEE Int. Conf. on Systems, Man and Cybernetics (SMC'99)*, pp 640-645. IEEE Press, Piscataway, NJ.
20. Morgan JJ and Kilgour AC. (1996) Personalising on-line information retrieval support with a genetic algorithm. In: Moscardini AO & Smith P. (Eds.) *Proc. of PolyModel 16: applications of artificial intelligence*, pp 142-149.
21. Paredis J. (2000) Co-evolutionary algorithms. In: Back T, Fogel DB and Michalewicz Z. (Eds) *Evolutionary computation 2: Advanced Algorithms and Operators*, pp 224-238. Institute of Physics Publishing, Bristol.
22. Poli R and Cagnoni S. (1997) Genetic programming with user-driven selection: experiments on the evolution of algorithms for image enhancement. *Genetic Programming 1997: Proc. 2nd Annual Conf.*, pp 269-277. Morgan Kaufmann, San Mateo, CA.
23. Salton G and Buckley C. (1998) Term-weighting approaches in automatic text retrieval. *Information Processing and Management 24*, 513-523. Reprinted in: Sparck Jones K and Willet P. (Eds.) *Readings in Information Retrieval*, pp 323-328. Morgan Kaufmann, San Mateo, CA.
24. Terano T and Ishino Y. (1998) Interactive genetic algorithm based feature selection and its application to marketing data analysis. In: Liu H and Motoda H. (Eds) *Feature Extraction, Construction and Selection*, pp 393-406. Kluwer, Boston.
25. Thuraisingham B. (2000) *Web Data Management and Electronic Commerce*. CRC Press.