Fundamenta Informaticae 35 (1998) 1–16
IOS Press

On evolutionary exploration and exploitation

A.E. Eiben*
Leiden University / CWI Amsterdam
Leiden / Amsterdam, The Netherlands
gusz@wi.leidenuniv.nl

C.A. Schippers*
Leiden University
Leiden, The Netherlands
adriaan@wi.leidenuniv.nl

Abstract. Exploration and exploitation are the two cornerstones of problem solving by search. The common opinion about evolutionary algorithms is that they explore the search space by the (genetic) search operators, while exploitation is done by selection. This opinion is, however, questionable. In this paper we give a survey of different operators, review existing viewpoints on exploration and exploitation, and point out some discrepancies between and problems with current views.

1. Introduction

Evolutionary algorithms (EA) belong to the family of stochastic generate-and-test search algorithms [28]. There are different types of EAs, the most common classification distinguishes Genetic Algorithms (GA), Evolution Strategies (ES) and Evolutionary Programming (EP), [4]. A fourth type of EA, Genetic Programming (GP) has grown out of GAs and is often seen as a sub-class of them. Besides the different historical roots and philosophy there are also technical differences between the three main streams in evolutionary computation. These differences concern the representation applied, the corresponding genetic search operators, the selection mechanism, and the role of self-adaptation. It is interesting to note that several generate-and-

*Address for correspondence: Leiden University, CWI Amsterdam, Leiden / Amsterdam, The Netherlands

test style search algorithms based on a not evolutionary paradigm, such as simulated annealing [1], tabu search [25]or local search can also be described in the evolutionary algorithm framework.

Like all generate-and-test search algorithms, EAs obtain their power from two sources: exploration and exploitation. The common opinion about EAs is that they explore the search space by the (genetic) search operators, while exploitation is done by selection. In this paper we question this opinion. We give an overview of existing viewpoints together with some drawbacks. Hereby we try to initiate debate and further investigations on the subject.

This paper is organized as follows. In Section 2 we describe different evolutionary search operators as used in different styles of EAs. In Section 3 we consider the issue of operator arity, that is mutation vs. recombination operators. The notions exploration and exploitation are discussed in Section 4 and we finally make some concluding remarks in Section 5.

2. Evolutionary search

Search operators can be roughly classified by their arity and their type, the latter meaning the data type or representation they are defined for. This data type or representation is one of the distinguishing features of the different styles of EAs. In traditional GAs binary representation is used, that is candidate solutions (individuals or chromosomes) are bit-strings with a fixed length L. The study of sequencing problems, such as routing and scheduling, has yielded order-based GAs, where each individual is a permutation, [51]. Parameter optimization problems with variables over continuous domains have led to real-valued (or floating point) GAs. Genetic Programming uses individuals that are tree structures, [35]. Accordingly, the search operators are defined for trees instead of strings. Real valued numerical optimization is the standard application area of Evolution Strategies [44]. Therefore, real-valued representation is the standard in ES. The original Evolutionary Programming scheme was applied to finite state machines, using appropriately defined operators, [21]. More recently, combinatorial and numerical optimization problems have been treated by EP, thus involving real-valued representation in the EP paradigm too.

As far as their arity is concerned, the operators used commonly in EAs are either unary or binary. Unary operators are based on analogies of asexual reproduction mechanisms in nature, while binary operators simulate sexual reproduction. The less common so-called multi-parent operators are n-ary; formally they are generalizations of mutation and (two-parent) crossover without a natural analogy.

In all evolutionary algorithms the selection mechanism considers the quality (fitness) of the individuals as the basis to make choices. Therefore almost all selection mechanisms are similar in this respect: the actual representation is irrelevant, only the fitness values are used and information on the "inside" of an individual is not. Furthermore, in EC usually the representation and the applied reproduction are seen as the critical factors in the successfulness of an EA. For these reasons, we omit a (representation related) overview of selection mechanisms, while we do present such a survey of search operators. In the remainder of this section we give an overview

of evolutionary search operators. Since there are numerous operators proposed and applied for specific purposes, this overview is inevitably incomplete. However, we do consider the most important ones of each EA-style (representation), sorted by operator arity.

2.1. Mutation

The most commonly used unary operator is mutation which is meant to cause a random and unbiased change in an individual, resulting in one new individual. In GAs for bit-representation a so-called mutation rate p_m is given as external control parameter. Mutation works by flipping each bit in the individual it is applied to independently, with probability p_m. Mutation rate is thus a parameter that is used while performing mutation in case of bit representation. The probability that an individual is actually mutated is a derived measure:

$$P[\text{individual is mutated}] = 1 - (1 - p_m)^L$$

In case of order-based representation swapping or shifting randomly selected genes within an individual can cause the random perturbation. In this case the mutation rate $p_m = P[\text{individual is}]$
mutated] is used *before* mutating to decide whether or not the mutation operator will be applied. For tree-structured individuals replacing a sub-tree (possibly one single node) by a random tree is appropriate. Mutation rate in GP is interpreted as the frequency of performing mutation, similarly to order-based GAs. It is remarkable that the general advice concerning the mutation rate in GP is to set it to zero, [35].

For the real-valued representation as used in ES, EP and lately in GAs the basic unit of inheritance, i.e. one gene, is a floating point number x_i. In ES and EP a separate mutation mechanism is used for each gene, i.e. object variable. Mutating a certain value x_i means perturbing that value with a random number drawn form a Gaussian distribution $N(0, \sigma_i)$ by

$$x_i' = x_i + \sigma_i \cdot N(0, 1).$$

That is, σ_i is the standard deviation, also called the mean mutation step size, concerning the variable (axis) x_i. It is essential that the σ_i's are not static external control parameters but are part of the individuals and undergo evolution. In particular, an individual contains not only the object variables x_i, but also the strategy parameters σ_i, resulting in $2 \cdot n$ genes for n object variables. (For this discussion we disregard the incorporation and adaptation of covariances.) In ES the mean step sizes are modified log-normally:

$$\sigma_i' = \sigma_i \cdot exp(\tau' \cdot N(0, 1) + \tau \cdot N_i(0, 1))$$

where $\tau' \cdot N(0, 1)$ and $\tau \cdot N_i(0, 1)$ provide for global, respectively individual control of step sizes, [4].

In Evolutionary Programming the so-called meta-EP scheme is introduced for adapting mean step sizes. It works by modifying σ's normally:

$$\sigma_i' = \sigma_i + \zeta \cdot \sigma_i \cdot N(0, 1)$$

where ζ is a scaling constant, [21]. Summarizing, the main differences between the GA and ES/EP mutation mechanisms are that

- in a GA every gene (bit) of every individual is mutated by the same mutation rate p_m, while ES/EP mutation is based on a different step size for every individual and every gene (object variable);
- p_m is constant during the evolution in a GA, while the step sizes undergo adaptation in ES/EP.

Let us note that varying the mutation rate over a single run was successfully applied also in GAs [20]. Cross-fertilization of ideas from ES and GAs has led to using individual mutation rates that undergo self-adaptation in a GA with binary representation, [3]. The basic idea is to extend each individual with extra bits that represent the individual's own mutation rate. Mutation, then, happens by first decoding these extra bits to the individual's own mutation rate p, mutating the extra bits by probability p, again decoding the (now mutated) extra bits to obtain the mutation rate p' which is finally applied to the 'normal' bits of this individual.

Recently, floating point representation is gaining recognition in GAs for numerical function optimization. Mutating a gene, then, can happen by uniform mutation, replacing x_i by a number x_i' drawn form its domain by a uniform distribution, or by non-uniform mutation, where

$$x_i' = \begin{cases} x_i + \Delta(t, UB_i - x_i) & \text{if a random digit is 0} \\ x_i - \Delta(t, x_i - LB_i) & \text{if a random digit is 1} \end{cases}$$

UB_i and LB_i being the upper, repectively lower bound of the domain and t the generation number (time counter), [38]. The function $\Delta(t, y)$ returns a value from $[0, y]$ getting closer to 0 as t increases. This mechanism yields a dynamically changing mutation step size. This resembles ES and EP, but differs from them in an important aspect. Namely, while in ES and EP step sizes are adapted by the evolutionary process itself, here the changes follow a previously determined schedule.

An interesting GA using bit representation combined with Gaussian mutation is presented in [30, 31]. The individuals are binary coded and undergo crossover acting on bits as usual. However, mutation is applied on the decoded values of the object variables x_i and not on their binary codes. Before mutation random numbers drawn from a Poisson distribution determine the object variables that are to be mutated. Mutation itself works by decoding that part of the binary code that represents a selected variable x_i to a real value and perturbing it by Gaussian noise from $N(0, \sigma_i)$, where σ_i is 0.1 times the maximum value of that gene. In the self-adapting version of this mechanism an extra gene, coded by a binary sequence, is added to the individual. During mutation the extra bits coding the extra gene are decoded to a value, this value σ is perturbed by an $N(0, \tau)$ Gaussian noise (τ is a control parameter), resulting in σ' and finally the object variables are mutated according to an $N(0, \sigma')$ distribution.

Relating the role of mutation to the explorative, respectively exploitative part of the search two different views are common. On the one hand, mutation is often seen as an explorative operator,

because it introduces new material, in an unbiased manner. On the other hand, it can also be seen as an exploitative operator, because it is conservative. That is, it causes only small changes, thus most of the old information, respectively genetic material, is preserved.

2.2. Crossover or recombination

As for sexual reproduction, the binary operator in GAs is called crossover, it creates two children from two parents. The crossover rate p_c prescribes the probability that a selected pair of would-be parents will actually mate, i.e. that crossover will be applied to them. It is thus used before crossover is applied. The interpretation of p_c is therefore not analogous to that of p_m in GAs with bit representation. There are two basic crossover mechanisms: n-point crossover and uniform crossover. The n-point crossover cuts the two parents of length L into $n+1$ segments along the randomly chosen crossover points (the same points in both parents) and creates $child_1$ by gluing the odd segments from the father and the even segments from the mother. In $child_2$ odd segments come from the mother and even segments come from the father. In the majority of GA applications 1-point crossover or 2-point crossover is used, higher n's are applied seldomly. Uniform crossover decides per bit whether to insert the bit value from the father or from the mother in $child_1$; $child_2$ is created by taking the opposite decisions. Order-based representation requires that each individual is a permutation. Applying n-point and uniform crossover to permutations may create children containing multiple occurrences of genes, i.e. children of permutations will not necessarily be permutations. This fact has led to the design of special order-based crossovers that do preserve the property of 'being a permutation', [51]. One example is the order crossover OX that resembles 2-point crossover. OX cuts the parents in three segments along two randomly chosen crossover points. $Child_1$ inherits the middle segment from $parent_1$ without modification. The remaining positions are filled with genes from $parent_2$ in the order in which they appear, and skipping genes already present in $child_1$. $Child_2$ is created analogously, reversing the parents' roles. Clearly, tree-based representation in GP requires special crossover operators too. The standard GP crossover works by randomly chosing one node in $parent_1$ and one in $parent_2$ independently. These nodes define a subtree in each parent, crossover exchanges these subtrees.

In Evolution Strategies there are different sexual recombination mechanisms all creating one child from a number of parents. The usual form of recombination, [4], produces one new individual $\bar{x}'$ from two parents $\bar{x}$ and $\bar{y}$ by

$$x_i' = \begin{cases} x_i \text{ or } y_i & \text{discrete recombination} \\ x_i + \chi \cdot (y_i - x_i) & \text{intermediate recombination} \end{cases}$$

Discrete recombination is pretty much like uniform crossover creating only one child. Intermediate recombination (where $\chi \in [0,1]$ is a uniform random variable) makes a child that is the weighted average of the two parents. Just like mutation, recombination is applied to the object variables (x_i's) as well as to the strategy parameters (σ_i's). Empirically, discrete recombination on object variables and intermediate recombination on strategy parameters gives best results.

Similar operators are also introduced for GAs with real valued representation applied to convex search spaces, [38]. These operators create two children of two parents. The so-called simple crossover choses a random crossover point k and crosses the parents after the k-th position applying a contraction coefficient a to guarantee that the offspring fall inside the convex search space. In particular, if $\bar{x}$ and $\bar{y}$ are the two parents then $child_1$ is

$$\langle x_1, \ldots, x_k, a \cdot y_{k+1} + (1-a) \cdot x_{k+1}, \ldots, a \cdot y_n + (1-a) \cdot x_n \rangle$$

and $child_2$ is obtained by interchanging x and y. Single arithmetic crossover effects only one gene in each parent. The children of $\bar{x}$ and $\bar{y}$ obtained by this mechanism are:

$$\langle x_1, \ldots, a \cdot y_k + (1-a) \cdot x_k, \ldots, x_n \rangle \text{ and } \langle y_1, \ldots, a \cdot x_k + (1-a) \cdot y_k, \ldots, y_n \rangle$$

where a is a dynamic parameter, drawn randomly from a domain calculated for each pair of parents in such a way that the children are in the convex space. Whole arithmetical crossover is somewhat similar to intermediate recombination in ES in that it creates the linear combinations

$$a \cdot \bar{x} + (1-a) \cdot \bar{y} \text{ and } a \cdot \bar{y} + (1-a) \cdot \bar{x}$$

as children of $\bar{x}$ and $\bar{y}$. Here again, the choice of a has to guarantee that the children are in the convex space.

A very particular feature within the EP paradigm is the total absence of sexual reproduction mechanisms. Mutation is the only search operator, and *'recombination in evolutionary programming is a nonissue because each solution is typically viewed as the analog of a species, and there is no sexual communication between species'*, cf. [21] p. 103. Indeed, this view justifies the omittance of recombination, however, this view is only justified by the human operator's freedom in taking any arbitrary perspective. Nevertheless, the fact that apparently all higher species apply sexual recombination indicates that it does have a competitive advantage in natural evolution.

Similarly to mutation, crossover can be seen as both explorative and exploitative. Crossover is an explorative operator, because it recombines the material of the parents into new configurations. In the meanwhile, it can be perceived as exploitative, because the material used is old.

2.3. Multi-parent recombination

Recombination operators with higher arities were proposed independently by a number of researchers, some already in the 60'ies [22, 11]. As part of the standard machinery, such operators were first used in ESs. The global form of recombination in ES produces one new individual $\bar{x}'$ by

$$x_i' = \begin{cases} x_i \text{ or } y_i & \text{global discrete recombination} \\ x_i + \chi_i \cdot (y_i - x_i) & \text{global intermediate recombination} \end{cases}$$

where $\bar{x}$ and $\bar{y}$ are two parent individuals selected at random from the whole population for each i anew and drawing $\chi_i \in [0,1]$ also happens for every i independently, [4]. Thus, although for

each component x_i' only two mating partners are consulted, the resampling mechanism causes that the child may inherit genes from more parents. This results in a higher mixing of the genetic information than the 2-parent versions.

In GAs there are two general multi-parent crosover mechanisms that are the straightforward generalizations of the traditional two-parent mechanisms. Scanning crossover and diagonal crossover were introduced in [14]. Scanning crossover generalizes uniform crossover, although creating only one child, by chosing one of the i-th genes of the n parents to be the i-th gene of the child. Formally,

$$x_i' = C(x_i^1, \ldots, x_i^n)$$

where C is a choice mechanism choosing one of the elements of a given set $\{u_1, \ldots, u_n\}$. The choice can be random, based on a uniform distribution (uniform scanning), or biased by the fitness of the parents (fitness based scanning). It can also be deterministic, based on the number of occurrences of the genes (occurrence based scanning). Diagonal crossover generalizes 1-point crossover by selecting $(n-1)$ crossover points and composing n children by taking the resulting n chromosome segments from the parents 'along the diagonals'.

The Gene Pool Recombination (GPR) in GAs was introduced in [40]. It was further studied and extended with a fuzzy mechanism in [49]. In GPR first a set of parents has to be selected, these form the so-called gene pool. Then two parents of an offspring are chosen randomly for each position, with replacement, from the gene pool (as opposed to ES, where they are chosen from the whole population). The offspring's allele is computed using any of the standard recombination schemes that work on two parents. GPR allows for theoretical analysis of GA convergence for infinite populations with binomial fitness distribution.

Note that multi-parent recombination intensifies the explorative properties of (two-parent) crossover. By using more parents there is more disruption than in the traditional case, hence the resemblence between offspring and a given parent is (on average) smaller. There is also a reason to see multi-parent recombination as more exploitative than (two-parent) crossover. Namely, more parents are utilized (exploited) when creating new individuals. Thus, the dual character (explorative and exploitative) of two-parent recombination applies even more to multi-parent operators.

3. The importance of mutation and recombination

The traditional views of the different styles of EAs on genetic operators are summarized in Table 1, after [4]. Triggered by the success of EP without recombination, as well as by 'introspective' motives, there is more and more research devoted to the usefulness of mutation and recombination in GAs and ESs. Although neither GAs nor EP were originally developed as function optimizers, [9, 21], such investigations are usually performed on function optimization problems.

Classical GA investigations tried to obtain a robust parameter setting concerning mutation rate and crossover rate [8, 26, 7]. The generally acknowledged good heuristic values for mutation

	ES	EP	GA
mutation	main operator	only operator	background operator
recombination	important for self-adaptation	none	main operator

Table 1 Role of mutation and recombination in different EA paradigms

rate and crossover rate are $1/L$ (L being the chromosome length), respectively 0.7-0.8. Recent studies, for instance [3, 20, 29, 34] provide theoretical and experimental evidence that decreasing mutation rate along the evolution is optimal. This suggests that mutation is important at the beginning of the search but becomes 'risky' at the end. In other words, exploration should gradually change into exploitation [1] In the meanwhile, the GA community lately acknowledges that mutation has a more important role than just (re)introducing absent or lost genes in the population.

As far as recombination is concerned, the main area of interest is traditionally the crossover's ability for combining and/or disrupting pieces of information, called schemata. Formally, a schema can be seen as a partial instantiation of the variables, i.e. a string based on a ternary alphabet $0, 1, \#$, where $\#$ means undefined or don't care. Investigations in for instance [10, 18, 45, 46], cumulated substantial knowledge on the disruptiveness of n-point and uniform crossover. The most important observation from [10] concerns the relationship between the disruptiveness and the power to recombine schemata. *'... if one operator is better than another for survival,* [i.e. less disruptive] *it is worse for recombination (and vice versa). This .. suggests very strongly that one cannot increase the recombination power without a corresponding increase in disruption.* Another important feature of crossover operators is the variety of different regions in the search space that can be sampled. In other words, the number of different chromosomes that can be created by applying an operator. Actually, it is not only which (new) chromosomes could be created, but also the probability distribution over this set. Clearly, *'exploration in the form of new recombinations comes at the cost of disruption'*, [18]. Recent research has thus shown that disruption is not necessarily disadvantageous. Nevertheless, no general conclusions on the relationship between the disruptiveness of genetic operators and the quality of the GA they are applied in could be established so far. Most probably this relation is (to some extent) problem dependent.

Besides comparisons of crossover operators, the relative importance, i.e. search power, of sexual recombination and asexual, unary operators have been investigated. The experiments

[1] Note that for recombination things are a bit different. The disruptiveness of mutation is not dependend on the individual it operates on. For recombination however this is indeed the case. Equivalent parents will yield children that are exact copies of those parents for most crossover operators and highly similar parents will yield children very similar to those parents. The fact that EAs often exhibit convergence, i.e. the indiviudals in the population become more and more similar, causes crossover to become gradually less disruptive during a run. In other words: the gradual change from explorative to exploitive comes about automatically for crossover.

reported in [43] show that mutation and selection are more powerful than previously believed. Nevertheless, it is observed that a GA with highly disruptive crossover outperforms a GA with mutation alone on problems with a low level of interactions between genes. It is also remarked that the power of an operator is strongly related to the selection mechanism. In [19] 'crossover's niche' is sought, i.e. problems where pair-wise mating has competitive advantages. This niche turns out to be non-empty, in the meanwhile the authors suspect that sexual recombination in GAs might be less powerful than generally believed. The usefulness of recombination is investigated on NK landscapes that allow gradual tuning of the ruggedness of the landscape in [32, 15]. The conclusion is that *'recombination is useless on uncorrelated landscapes, but useful when high peaks are near one another and hence carry mutual information about their joint locations in the fitness landscape'.* The view presented in [45] relativizes the mutation-or-crossover battle by noting that both operators serve another purpose. *'Mutation serves to create random diversity in the population, while crossover serves as an accelerator that promotes emergent behavior from components.'* Additionally, crossover is useful for maximizing the accumulated payoff, while it can be harmful if optimality is sought.

Based on a comparison of EP and GA in [23] it is argued that crossover does not have any competitive advantage over mutation. Critiques on this investigation, for instance in [43], initiated more extensive comparisons of GAs and EP on function optimization problems. The results in [24] show a clear advantage of EP and make the authors conclude that *'no consistent advantage accrues from representing real-valued parameters as binary strings and allowing crossover to do within-parameter search.'* These results seem to suggest that a minimal operator arity is the best option in an EA. The experimental investigations on multi-parent recombination [13, 15, 16, 12, 11] seem to contradict this. Namley, it turns out that increasing the operator arity can be advantageous for very different operators, under very different circumstances.

At the moment there is a lot of effort being payed to establishing the (dis)advantages of sexual, respectively asexual reproduction in simulated evolution. So far there is no generally valid judgement, and it is possible that the merits of any type of recombination are context and goal dependent.

Considering Sections 2 and 3, it is clear that although the search operators mutation and (multi-parent) recombination are crucial in EAs, they are still not well understood. Understanding the operators' role with respect to exploration and exploitation ("what are they doing") may help in determining which operator is preferable under what circumstances.

4. On exploration and exploitation

During the years of EA history the terms exploration and exploitation have been playing an important role in describing the working of an algorithm. In particular, search operators (mutation and recombination) and selection have been characterized by their contribution to the explorative and exploitative aspects of the search. In this section we present an overview of existing approaches to these issues. Let us note that very few papers explicitely investigate

exploration and exploitation. Most authors leave their definitions implicit and use the intuitive meaning of the concepts to explain the working of EAs.

Although exploration and exploitation occur in all types of EAs, most authors restrict their discussion to bit-string representation or real-valued representation, the first one being dominant. Among the ten papers we review here seven consider bit-strings only, two real-valued only, and one mentions them both.

We can distinguish three levels at which the phenomena exploration and exploitation, occur: at individual level, at sub-individual level (combinations of more than one gene, i.e. schemata in bit-string GAs, or masks in real-valued EAs) and at single gene level. Let us clarify these levels a bit. To start with the individual level, at this level indiviuals are atomic, i.e. we can not take a look into a particular indiviual. Available information is limited, for instance to measuring the distance between two indiviudals (e.g. Hamming distance). At the second level each individual is perceived as an instance of 2^l schemata [2], where l is the length of the individual. Membership of a schema is the basic property that can be inherited by offspring. The third level is a special case of the second one. Values of a single gene form the inheritable properties here, i.e. schemata of length one. If we do not restrict ourselfs to traditional GAs, bias of an allelic value towards the corresponding allele of the parent can also be seen as inheriting a property.

An example of the first level is given in [48] pp.239, where exploitation of "the neighborhood of the best solution" is discussed. The "common string patterns" considered in [41] pp. 132, form an illustration of our second level. The lowest level can be illustrated by a real-valued EP that performs a Gaussian perturbation of a value (allele), thereby exploiting it.

Selection is commonly seen as *the* source of exploitation, while exploration is attributed to the operators mutation and recombination: "Selection according to fitness is the source of exploitation [...] mutation and crossover operators are the sources of exploration" [18]. This implies that usage of material is identified with exploitation. A more refined vision restricts exploitation to good use [3] of information [47]. As Beyer states "Exploitation is defined in this context as the ability of an algorithm to step into the direction of desired improvement." [6]. Handa et al. in [47] describe an algorithm with a seperate and explicitly exploitive part for "effective utilization of symbolised information".

Another widely spread opinion is that exploration and exploitation are opposite forces. Many authors state the importance of finding a good balance between the two: "the effectiveness of the GA depends upon the appropriate mix of exploration and exploitation" [18], "From this emerges a focus on the exploitation/exploration tradeoff", [19], pp.10, "[...] to balance the degrees of exploration and exploitation [...]", [48] pp. 238 and "This reemphasizes the need for a balance between exploration and exploitation", [10] pp. 22. Eshelman, Schaffer and Caruana are rather explicit when they state: "Increased exploitation by selection leads to decreased exploration by crossover.", [18], pp. 11. But later Eshelman and Schaffer put things in perspective when differentiating between "exploratory trials" and "exploitive trials" [4], [19], pp. 10.

[2] For clarity we restrict ourselfs to the traditional GA here, i.e. bitstring representation is assumed.

[3] We will call "good use" utilization.

[4] A trial is the generation and evaluation of a new point in the search space. The trial is exploitive with

In the following table we give a summary of viewpoints represented in the papers of our review. The second column shows which representation is considered. The third column shows whether or not exploitation is identified by usage of information, whereas the fourth column tells wether the contribution of the operators (mutation and recombination) is purely explorative in nature. In the last column we indicate whether the paper in question considers exploitation and exploration as opposite forces. Let us note that since many papers do not explicitely discuss these issues, we are forced to interpret the text and destill the authors' opinions on these issues

paper	representation	level	usage identified with exploitation	the operators are purely explorative	opposing forces
[41]	bit+float	2	yes	yes	yes
[50]	bit	2	-	-	-
[19]	bit	2	yes	no	yes
[10]	bit	2	-	no	yes
[18]	bit	2	yes	yes	yes
[47]	bit	2	no	-	-
[36]	bit	2	yes	yes	-
[48]	float	1	yes	-	yes
[19]	bit	2	-	no	-
[6]	float	1	no	no	no

Table 2 Summary of viewpoints on exploration and exploitation ('-' means that we were not able to infer the authors point of view on the subject)

It is clear that there is no general consensus on these matters; several authors represent contradicting views. This leads to a number of questions, respectively hypotheses, to be investigated.

1. Using (existing) information equals exploitation. This would imply that even the use of bad material (information), or bad use of good information is seen as exploitation. Of course it might be possible to use bad material to identify unpromissing areas in the search space. These areas could then be avoided. Alternatively, exploitation could be restricted to good use of good information.
2. Selection is *the* source of exploitation. This is more or less a corollary of 1.
3. The operators mutation and recombination are purely explorative. This might seem an evident truth at first; after all, creating variation is the goal of these operators. But on the other hand exploiting a schema could be defined as exploring the corresponding hyperplane. *New* points within this hyperplane are generated by mutation and recombination, when

respect to a schema, if and only if that schema whas present in a parent (and not in his mate) and survives in the offspring. Otherwise it is exploratory with respect to that schema.

the schema is not disrupted. Eshelman and Schaffer even call these events "exploitive trials" [19].

4. Exploitation and exploration are opposing forces that need to be balanced. This too, might seem evident at first, but this is only so when exploitation is identified with usage. At the schemata level, for instance, maximizing the number of schemata used, is equal to cloning an existing individual. But suppose that all good schemata of this individual are in its left half and another individual has all its good schemata in his right half. If exploitation would be identified by use of good material, we could increase exploitation by recombining the good halfs of both parents in the new individual, thereby simultatiously increasing exploration.

5. What information (on alleles, schemata, individuals) is available and which part of this information is actually exploited? This is a complicated issue because it is highly problem dependent. When there is no epistatic interaction for example, a good combination of alleles gives a lot of information: sub-combinations can be used in new indiviudals to enhance their fitness. When all of the genes in the combination are in epistatic interaction however, the only information we have, is that the entire combination enhances the fitness of an individual.

6. What is the influence of representation on all these issues? Battle and Vose showed in [5] that given any function f on bitstrings there exists an invertable mapping (1-1 correspondence) of bitstrings that basically transforms f to countings ones. This proves that representation has a major influence on what constitutes information, for in general it is not true that the number of ones in a bitstring and the fitness of that bitstring have a significant statistical corrolation.

7. Sometimes alleles (or combinations of these values) are exploited by generating new indiviuals containing these alleles. In other cases exploitation is more subtle and the values of genes are only biased by the alleles of the ancestor. Further investigations on this difference seem interesting.

8. What can be said on the difference in exploration and exploitation between different types of EAs?

9. Exploitation in EAs is most of the time done very implicitly. The fitness of individuals is used to determine wether or not an individual should be exploited. After that it is hoped and believed (schema theorem) that the algorithm will somehow put the individual to good use. In [48] some research on an algorithm with explicit exploitation is described. May be explicit exploitation could enhance the performance of EAs, but intuitively it seems to be in conflict with the original idea of an elegant, simple, and good algorithm.

10. How long can or should material be exploited? This is another interesting question. Normally it is left to the (stochastic!) selection mechanism and based on the fitness only.

5. Concluding remarks

Evolutionary search is often characterized by fundamental dualities, such as creating and reducing diversity in the population through the application of reproduction (including mutation and recombination) and selection. The notions of exploration and exploitation (which are more general, since they occur in other forms of search too), are often related to this duality. In this paper we gave a survey of the most commonly used representations and reproduction operators, and discussed their role in the search for a solution. We omitted a survey of selection mechanisms, for reasons discussed in section 2. Furthermore, we considered existing views on exploration and exploitation, relating these notions to the search operators. This overview reveals that there is no generally accepted perception on exploration and exploitation in evolutionary computing. There are, however, a number of important research issues, listed in the previous section. Clearly, intensive research is needed for a deeper understanding of the fundamentals of evolutionary search processes.

References

[1] E. Aarts and J. Korst. *Simulated Annealing and Boltzmann Machines.* Wiley and Sons, 1989.

[2] T. Bäck, D. Fogel, and Z. Michalewicz, editors. *Handbook of Evolutionary Computation.* Institute of Physics Publishing Ltd, Bristol and Oxford University Press, New York, 1997.

[3] Th. Bäck. The interaction of mutation rate, selection, and self-adaptation within a genetic algorithm. In Männer and Manderick [37], pages 85–94.

[4] Th. Bäck and H.-P. Schwefel. An overview of evolutionary algorithms for parameter optimization. *Evolutionary Computation*, 1(1):1–23, 1993.

[5] D.L. Battle and M.D. Vose. Isomorphisms of genetic algorithms. In Grefenstette [27], pages 242–251.

[6] H-G. Beyer. On the "explorative power" of ES/EP-like algorithms. In N. Saravanan, D. Wagen, and A.E. Eiben, editors, *Proceedings of the 7th Annual Conference on Evolutionary Programming*, Lecture Notes in Computer Science. Springer, Berlin, 1998. in press.

[7] R.A. Caruana, L.J. Eshelman, J.D. Schaffer, and R. Das. A study of control parameters affecting the online performance of genetic algorithms. In Schaffer [42], pages 51–60.

[8] K.A. De Jong. *An analysis of the behavior of a class of genetic adaptive systems.* Doctoral dissertation, University of Michigan, 1975.

[9] K.A. De Jong. Are genetic algorithms function optimizers? In Männer and Manderick [37], pages 3–13.

[10] K.A. De Jong and W.M. Spears. A formal analysis of the role of multi-point crossover in genetic algorithms. *Annals of Mathematics and Artificial Intelligence*, 5:1–26, 1992.

[11] A.E. Eiben. Multi-parent recombination. In Bäck et al. [2]. Section C3.3.7, to appear in the 1st supplement.

[12] A.E. Eiben and Th. Bäck. An empirical investigation of multi-parent recombination operators in evolution strategies. *Evolutionary Computation*, 5(3):347–365, 1997.

[13] A.E. Eiben, C.H.M. van Kemenade, and J.N. Kok. Orgy in the computer: Multi-parent reproduction in genetic algorithms. In Morán et al. [39], pages 934–945.

[14] A.E. Eiben, P-E. Raué, and Zs. Ruttkay. Genetic algorithms with multi-parent recombination. In Y. Davidor, H.-P. Schwefel, and R. Männer, editors, *Proceedings of the 3rd Conference on Parallel Problem Solving from Nature*, number 866 in Lecture Notes in Computer Science, pages 78–87. Springer-Verlag, 1994.

[15] A.E. Eiben and C.A. Schippers. Multi-parent's niche: n-ary crossovers on NK-landscapes. In H.-M. Voigt, W. Ebeling, I. Rechenberg, and H.-P. Schwefel, editors, *Proceedings of the 4th Conference on Parallel Problem Solving from Nature*, number 1141 in Lecture Notes in Computer Science, pages 319–328. Springer, Berlin, 1996.

[16] A.E. Eiben and C.H.M. van Kemenade. Diagonal crossover in genetic algorithms for numerical optimization. *Journal of Control and Cybernetics*, 26(3):447–465, 1997.

[17] L.J. Eshelman, editor. *Proceedings of the 6th International Conference on Genetic Algorithms*. Morgan Kaufmann, 1995.

[18] L.J. Eshelman, R.A. Caruana, and J.D. Schaffer. Biases in the crossover landscape. In Schaffer [42], pages 10–19.

[19] L.J. Eshelman and J.D. Schaffer. Crossover's niche. In S. Forrest, editor, *Proceedings of the 5th International Conference on Genetic Algorithms*, pages 9–14. Morgan Kaufmann, 1993.

[20] T. Fogarty. Varying the probability of mutation in the genetic algorithm. In Schaffer [42], pages 104–109.

[21] D.B. Fogel. *Evolutionary Computation*. IEEE Press, 1995.

[22] D.B. Fogel, editor. *Evolutionary Computation: the Fossile Record*. IEEE Press, 1998.

[23] D.B. Fogel and J.W. Atmar. Comparing genetic operators with Gaussian mutations in simulated evolutionary processes using linear systems. *Biological Cybernetics*, 63:111–114, 1990.

[24] D.B. Fogel and L.C. Stayton. On the effectiveness of crossover in simulated evolutionary optimization. *Biosystems*, 32:171–182, 1994.

[25] F. Glover. Tabu search and adaptive memory programming — advances, applications, and challenges. In R.S. Barr, R.V. Helgason, and J.L. Kennington, editors, *Interfaces in Computer Science and Operations Research*, pages 1–75. Kluwer Academic Publishers, Norwell, MA, 1996.

[26] D.E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, 1989.

[27] J.J. Grefenstette, editor. *Proceedings of the 2nd International Conference on Genetic Algorithms and Their Applications*. Lawrence Erlbaum Associates, 1987.

[28] J. Heitkotter and D. Beasley. *The Hitch-Hiker's Guide to Evolutionary Computation*. Number Issue 5.3. FAQ for comp.ai.genetic, October 1997.
http:// research.Germany.EU.net:8080/encore/www.

[29] J. Hesser and R. Männer. Towards an optimal mutation probability for genetic algorithms. In H.-P. Schwefel and R. Männer, editors, *Proceedings of the 1st Conference on Parallel Problem Solving from Nature*, number 496 in Lecture Notes in Computer Science, pages 23–32. Springer-Verlag, 1991.

[30] R. Hinterding. Gaussian mutation and self-adaptation for numeric genetic algorithms. In IEEE [33], pages 384–389.

[31] R. Hinterding, H. Gielewski, and T.C. Peachey. The nature of mutation in genetic algorithms. In Eshelman [17], pages 65–72.

[32] W. Hordijk and B. Manderick. The usefulness of recombination. In Morán et al. [39], pages 908–919.

[33] *Proceedings of the 2nd IEEE Conference on Evolutionary Computation*. IEEE Press, 1995.

[34] B. A. Julstrom. What have you done for me lately? Adapting operator probabilities in a steady-state genetic algorithm. In Eshelman [17], pages 81–87.

[35] J.R. Koza. *Genetic Programming*. MIT Press, 1992.

[36] Ting Kuo and Shu-Yuen Hwang. A genetic algorithm with disruptive selection. In Schaffer [42], pages 65–69.

[37] R. Männer and B. Manderick, editors. *Proceedings of the 2nd Conference on Parallel Problem Solving from Nature*. North-Holland, 1992.

[38] Z. Michalewicz. *Genetic Algorithms + Data structures = Evolution programs*. Springer, Berlin, 3rd edition, 1996.

[39] F. Morán, A. Moreno, J. J. Merelo, and P. Chacón, editors. *Advances in Artificial Life. Third International Conference on Artificial Life*, volume 929 of *Lecture Notes in Artificial Intelligence*. Springer, Berlin, 1995.

[40] H. Mühlenbein and H.-M. Voigt. Gene pool recombination in genetic algorithms. In I.H. Osman and J.P. Kelly, editors, *Meta-Heuristics: Theory and Applications*, pages 53–62, Boston, London, Dordrecht, 1996. Kluwer Academic Publishers.

[41] Xiaofeng Qi and Francesco Palmieri. The diversification role of crossover in the genetic algorithms. In Schaffer [42], pages 132–137.

[42] J.D. Schaffer, editor. *Proceedings of the 3rd International Conference on Genetic Algorithms*. Morgan Kaufmann, 1989.

[43] J.D. Schaffer and L.J. Eshelman. On crossover as an evolutionary viable strategy. In R.K. Belew and L.B. Booker, editors, *Proceedings of the 4th International Conference on Genetic Algorithms*, pages 61–68. Morgan Kaufmann, 1991.

[44] H.-P. Schwefel. *Evolution and Optimum Seeking*. Wiley, New York, 1995.

[45] W.M. Spears. Crossover or mutation? In L.D. Whitley, editor, *Foundations of Genetic Algorithms - 2*, pages 221–238. Morgan Kaufmann, 1993.

[46] G. Syswerda. Uniform crossover in genetic algorithms. In Schaffer [42], pages 2–9.

[47] Shigeyoshi Tsutsui, Ashish Ghosh, David Corne, and Yoshiji Fujimoto. Coevolutionary genetic algorithm with effective exploration and exploitation of useful schemata. In *The Fourth International Conference on Neural Information Processin*, pages 424–427, 1997.

[48] Shigeyoshi Tsutsui, Ashish Ghosh, David Corne, and Yoshiji Fujimoto. A real coded genetic algorithm with an explorer and an exploiter populations. In Th. Bäck, editor, *Proceedings of the 7th International Conference on Genetic Algorithms*, pages 238–245. Morgan Kaufmann, 1997.

[49] H.-M. Voigt and H. Mühlenbein. Gene pool recombination and utilization of covariances for the Breeder Genetic Algorithm. In IEEE [33], pages 172–177.

[50] M.D. Vose and G.E. Liepens. Schema disruption. In Grefenstette [27], pages 237–242.

[51] D. Whitley. Permutations. In Bäck et al. [2], pages B3.3:14–B3.3:20.