Evolution in Time and Space - The Parallel Genetic Algorithm

Heinz Mühlenbein
GMD Schloss Birlinghoven
D-5205 Sankt Augustin1

Abstract

The parallel genetic algorithm (PGA) uses two major modifications compared to the genetic algorithm. Firstly, selection for mating is distributed. Individuals live in a 2-D world. Selection of a mate is done by each individual independently in its neighborhood. Secondly, each individual may improve its fitness during its lifetime by e.g. local hill-climbing. The PGA is totally asynchronous, running with maximal efficiency on MIMD parallel computers. The search strategy of the PGA is based on a small number of active and intelligent individuals, whereas a GA uses a large population of passive individuals. We will investigate the PGA with deceptive problems and the traveling salesman problem. We outline why and when the PGA is succesful. Abstractly, a PGA is a parallel search with information exchange between the individuals. If we represent the optimization problem as a fitness landscape in a certain configuration space, we see, that a PGA tries to jump from two local minima to a third, still better local minima, by using the crossover operator. This jump is (probabilistically) successful, if the fitness landscape has a certain correlation. We show the correlation for the traveling salesman problem by a configuration space analysis. The PGA explores implicitly the above correlation.

Keywords: parallel optimization, parallel genetic algorithm, deceptive problems, traveling salesman

1 Introduction

Random search methods based on evolutionary principles have been proposed in the 60's. They did not have a major influence on mainstream optimization. We

believe that this will change. The unique power of evolutionary algorithms shows up with parallel computers. An example is our parallel genetic algorithm PGA introduced in 1987 [MGSK87]. It runs especially efficiently on parallel computers. Moreover our research indicates that parallel searches with information exchange between the searches are often better than a single search. Thus the PGA is a truly parallel algorithm which combines the hardware speed of parallel processors and the software speed of intelligent parallel searching.

We have successfully applied the PGA to a number of problems, including function optimization [MSB91] and combinatorial optimization. In this paper we try to explain the search strategy of the PGA in discrete optimization problems. Numerical results for the traveling salesman problem TSP, the quadratic assignment problem and the graph partitioning problem can be found in [GS89], [Müh89] and [vLM91].

In all these applications we have used the same algorithm with only slight modifications. We have taken the largest published problem instances known to us. The PGA found solutions, which are comparable or even better than any other solution found by other heuristics. This is proof by experimental result. We hope that future researchers will start where we have left off. The evaluation of heuristics for difficult combinatorial optimization problems has to be done with large problems. The results on small toy problems cannot be extrapolated to large problems.

The most difficult part of a random search method is to explain why and when it will work. We will do this analysis for the PGA with the deceptive problems and the TSP.

Throughout the paper we will use biological terms. We believe, that this is a source of inspiration and helps to understand the PGA intuitively. We do not claim that modelling natural evolution will automatically lead to a good optimization method.

The outline of the paper is as follows. In section 2 the problem of a parallel search with linkage is introduced. Evolutionary algorithms and genetic algorithms are reviewed in section 3. The importance of a spatial population structure on evolution is discussed in section 4. The next four sections show the influence of hill-climbing and crossing-over on the PGA. Analytical and numerical results are presented for deceptive problems proposed by Goldberg [Gol89b]. The more difficult TSP is discussed in sections 9 and 10. First the influence of the hill-climbing strategy is shown, then crossing-over is discussed by a configuration space analysis.

2 Parallel search and optimization

In this paper we consider the following problem:

OPT 1 *P: Given a function $F : X \mapsto R$, where X is some metric space. Let S be a subspace of X. We seek a point x in S which optimizes F on S or at least yields an acceptable approximation of the suprenum of F on S.*

Many optimization methods have been proposed for the solution of this problem. We will investigate parallel optimization methods. A parallel optimization method

of parallelism N is characterized by N different search trajectories, which are performed in parallel. It can be described as follows

$$x_i^{t+1} = G_i(x_1^t, ..x_N^t, F(x_1^t), ..F(x_N^t))i = 1, ..., N \quad (1)$$

The mapping $G = (G_1, ...G_N)$ describes the linkage or information exchange between the parallel searches. If the N searches are independent of each other we just have

$$x_i^{t+1} = G(x_i^t, F(x_i^t)) \quad (2)$$

A parallel search method which combines the information from two searches can be described as follows

$$x_i^{t+1} = G_i(x_{i-1}^t, x_i^t, F(x_{i-1}^t), F(x_i^t))i = 1, ..., N \quad (3)$$

The basic questions of parallel search methods can now be stated

- Are N parallel searches of time complexity t as efficient as a single search of time complexity $N * t$?
- Are N linked searches more efficient than N independent searches?
- How should the linkage be done?

In order to understand these questions intuitively, we leave the abstract mathematical description and turn to a natural search metaphor. The advantage of using a metaphor is that it leads to a qualitative understanding of the problem and the algorithm.

In this paper we use the following simple metaphor: The search is done by N active individuals, x_i describes the position of individual i and $F(x_i)$ its current value, representing the height in an unknown landscape. How should the population of individuals search the unknown landscape?

Many different models are possible. The individuals could be geographers who want to find the highest mountain. There is fog all over the place. The geographers can communicate with each other by broadcasting their information. How should they best do it? Many different strategies can be thought of. In a search algorithm we would then try to mimic their behavior.

In this paper, we will investigate search algorithms which mimic evolutionary adaptation found in nature. Each individual is identified with an animal, which searches for food and produces offspring. In evolutionary algorithms, $F(x_i)$ is called the fitness of individual i, x_i^{t+1} is an offspring of x_i^t, and G is called the selection schedule.

3 Evolutionary algorithms and genetic algorithms

A survey of search strategies based on evolution has been done in [MGSK88]. We recall only the most important ones. A generic evolutionary algorithm can be described as follows

Evolutionary algorithm

STEP1: Create an initial population of size $M = N * O$

STEP2: Compute the Fitness $F(x_i) i = 1, \ldots, M$

STEP3: Select N individuals according to some selection schedule

STEP4: Create O offspring of each of the N individuals by small variation

STEP5: If not finished, return to STEP2

Variants of this algorithm have been invented by many researchers, see for example [Rec73], [Sch81]. An evolutionary algorithm is a random search which uses selection and variation. The linkage of the parallel searches is only implicit by the selection schedule. Searches with bad results so far are abandoned and new searches are started in the neighborhood of more promising searches.

In biological terms, evolutionary algorithms model natural evolution by asexual reproduction with mutation and selection. Search algorithms which model sexual reproduction are called genetic algorithms. They were invented by Holland [Hol75]. Recent surveys can be found in [Gol89a] and [Sch89].

Genetic Algorithm

STEP0: Define a genetic representation of the problem

STEP1: Create an initial population $P(0) = x_1^0, ..x_N^0$

STEP2: Compute the average fitness $\overline{F} = \sum_i^N F(x_i)/N$. Assign each individual the normalized fitness value $F(x_i^t)/\overline{F}$

STEP3: Assign each x_i a probability $p(x_j, t)$ proportional to its normalized fitness. Using this distribution, select N vectors from $P(t)$. This gives the set $S(t)$

STEP4: Pair all of the vectors in $S(t)$ at random forming $N/2$ pairs. Apply crossover with probability p_{cross} to each pair and other genetic operators such as mutation, forming a new population $P(t+1)$

STEP5: Set $t = t + 1$, return to STEP2

In the simplest case the genetic representation is just a bitstring of length n, the "chromosome". The positions of the strings are called "locus" of the chromosome. The variable at a locus is called "gene", its value "allele". The set of chromosomes is called the "genotype"" which defines a "phenotype" (the individual) with a certain fitness. We will later show with examples why and when crossover guides the search.

A genetic algorithm is a parallel random search with centralized control. The centralized part is the selection schedule. The selection needs the average fitness of all individuals. The result is a highly synchronized algorithm, which is difficult to implement efficiently on parallel computers.
In our parallel genetic algorithm, we use a distributed selection scheme. This is achieved as follows. Each individual does the selection by itself. It looks for a partner in its neighborhood only. The set of neigborhoods defines a spatial population structure.

Our second major change can now easily be understood. Each individual is active and not acted on. It may improve its fitness during its lifetime by performing a local search.
A generic parallel algorithm can be described as follows

Parallel genetic algorithm

STEP0: Define a genetic representation of the problem

STEP1: Create an initial population and its population structure

STEP2: Each individual does local hill-climbing

STEP3: Each individual selects a partner for mating in its neighborhood

STEP4: An offspring is created with genetic crossover of the parents

STEP5: The offspring does local hill-climbing. It replaces the parent, if it is better than some criterion (acceptance)

STEP6: If not finished, return to STEP3.

It has to be noted, that each individual may use a different local hill-climbing method. This feature will be important for problems, where the efficiency of a particular hill climbing method depends on the problem instance.

In the terminology of section 2, we can describe the PGA as a parallel search with a linkage of two searches. The linkage is done probabilistically constrained by the neighborhood. The information exchange within the whole population is a diffusion process because the neighborhoods of the individuals overlap.

In a parallel genetic algorithm, all decisions are made by the individuals themselves. Therfore the PGA is a totally distributed algorithm without any central control.

There have been several other attempts to implement a parallel genetic algorithm. Most of the algorithms run k identical standard genetic algorithms in parallel, one run per processor. They differ in the linkage of the runs. Tanese [Tan89] introduces two *migration* parameters: the *migration interval*, the number of generations between each migration, and the *migrationrate*, the percentage of individuals selected for migration. The subpopulations are configured as a binary n-cube. A similar approach is done by Pettey et al. [PS89]. In the implementation of Cohoon et al. [CHMR87] it is assumed that each subpopulation is connected to each other. The algorithm from Manderick et al. [MS89] has been derived from our PGA. In this algorithm the individuals of the population are placed on a planar grid and selection and crossing-over are restricted to small neighborhoods on that grid.

All but Manderick's algorithm use subpopulations that are densely connected. We will show in the next section why restricted connections like a ring are much better for the genetic algorithm. All the above parallel algorithms do not use hill-climbing, which is one of the most important parts of our PGA.

An extension of the PGA, where subpopulations are used instead of single individuals, has been described in [MSB91]. This algorithm out-performs the standard GA by far in the case of function optimization. It is also a better search method than most of the standard mathematical methods.

We will explain in the next sections the three important parts of the search strategy of the PGA - the spatial population structure, hill-climbing and crossing-over.

4 Diversification by a spatial population structure

Genetic algorithms suffer from the problem of premature convergence. In order to solve this problem, many genetic algorithms enforce diversification explicitly, violating the biological metaphor. A popular method is to accept an offspring only if it is more than a certain factor different from all the members of the population.

The PGA tries to introduce diversification more naturally by a spatial population structure. Fitness and mating is restricted to neighborhoods called *demes*. This name has been introduced in population genetics, where it is well known that a population with a spatial structure has more variety than a panmictic population. The importance of this fact on evolution has been however highly controversially discussed.

Wright [Wri32] has argued that the best way to avoid being hung up on a low fitness peak is to have the population broken up into many nearly isolated subpopulations. Wright's theory has two phases. In the first phase of the evolution, the allele frequencies drift to some extent in each subgroup. One subgroup might by chance drift into a set of gene frequencies that correspond to a higher peak. Then the second phase sets in. This subgroup now has a higher fitness than other subgroups and will tend to displace them until eventually the whole population has the new, favorable gene combination. Then the whole process starts again.

Fisher [Fis58], in contrast, argued that no such theory is needed. In a highly multidimensional fitness surface, the peaks are not very high and are connected by fairly high ridges, always shifting because of environmental changes. According to Fisher, the analogy is closer to waves and troughs in an ocean than in a static landscape. Alleles are selected because of their average effects, and a population is unlikely to ever be in such a situation that it can never be improved by direct selection based on additive variance.

The difference between these two views is not purely mathematical, but physiological. Does going from one favored combination of alleles to another often necessitate passing through genotypes that are of lower fitness? Fisher argued that evolution typically proceeded in a succession of small steps, leading eventually to large differences by the accumulation of small ones. According to this view, the most effective population is a large panmictic one in which statistical fluctuations are slight and each allele can be fairly tested in combination with many others alleles. According to Wright's view, a more favorable structure is a large population broken up into subgroups, with migration sufficiently restricted (less than one migrant per generation) and size sufficiently small to permit appreciable local differentiation.

Three different mathematical models for spatially structured populations have been proposed

- the island model
- the stepping stone model

- the isolating by distance model

In the island model, the population is pictured as subdivided into a series of randomly distributed demes among which migration is random.

The stepping-stone model deals with discrete demes, separated into distinct subpopulations. Migration takes place between neighboring demes only.

The isolation by distance model treats the case of continuous distribution where effective demes are isolated by virtue of finite home ranges (neighborhoods) of their members. For mathematical convenience it is asssumed that the position of a parent at the time it gives birth relative to that of its offspring when the latter reproduces is normally distributed.

Felsenstein [Fel75] has shown that in many cases the above models lead to unrealistic clumping of individuals and concluded, that they are biologically irrelevant. There have been many attempts to investigate spatial population structures by computer simulations, but they did not have a major influence.

The issue raised by Wright and Fisher is still not settled. Because of the difficulty of solving the problem by mathematical analysis, population genetics has unfortunately lost interest in the problem. A good survey of the different population models can be found in Felsenstein [Fel76].

The PGA implements Wright's model. The two phases of Wright's theory can actually be observed. But the second phase performs differently. The biggest changes of the population occur at the time after migration between the subpopulations. Recombinations between immigrants and native individuals will occasionally lead to higher peaks which were not found by any of the subpopulations during isolation.

The creative forces of evolution take place at migration and few generations afterwards. Wright's argument that better peaks are found just by chance in small subpopulations does not capture the essential facts.

The above described phases can be easily shown in the application function optimization. We take as example searching the global minimum of a two dimensional function. The function is called Shekel's foxholes or DeJong's function F5. It has 25 local minima. The values of the minima are in the bottom row 1,2,3,4,5, in the second row 6,7,.. and so on. The value at the plateau is 500. In this example we have a fitness landscape which violates Fisher's assumption. To get from one valley to another one the population has to climb to the plateau, there are no other connections (because it is a minimization problem the valleys take over the role of the peaks in maximization).

We try to solve this problem with four subpopulations of 10 individuals each. Migration takes place after 10 generations. In figure 1 the progress of two subpopulations is shown. The number denotes the generation number. After generation ten subpopulation one is concentrated in four valleys, subpopulation two in two valleys. Subpopulation one has the y value of the optimum, subpopulation 2 the x value. After migration and recombination, subpopulation one expands into five valleys, subpopulation two into four valleys. Subpopulation one has now the genetic material to discover the lowest valley by recombination. At generation 12 subpopulation 1 has discovered the second lowest valley and it finds the global minimum at genera-

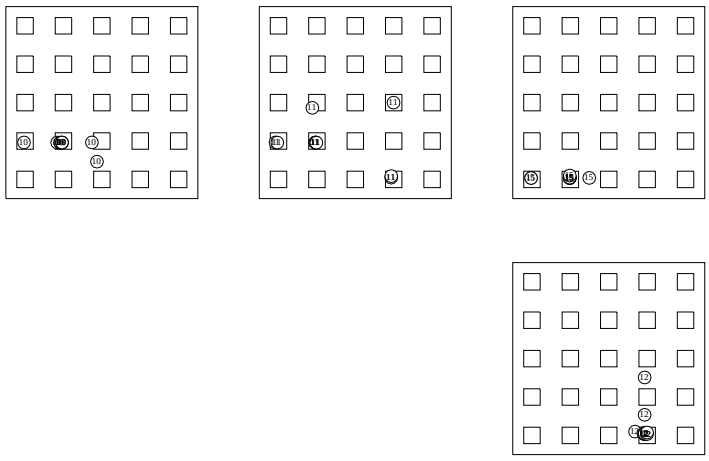

Figure 1: Evolution of two subpopulations

tion 15. Subpopulation one is now concentrated in the two lowest valleys. A detailed discussion about the PGA in function optimization can be found in [MSB91].

We summarize the most important conclusion of this section.

Subpopulations isolated for a certain time keep the diversity of the population high. After migration new promising areas can be discovered by crossing-over.

In most of our applications we have used a population structure called ladder. Our ladder is circular, both ends are closed. Why not use a more obvious structure like a torus or a still more densely connected structure? The answer is simple. In a ladder the variety of the population remains higher than in torus. In a torus of size n*n it takes $O(n)$ steps for an item of information to propagate to all places, in a ladder of the same size it takes $O(n^2)$ steps.

We now turn to the analysis of crossing-over.

5 The schema theorem revisited

Many researchers in the genetic algorithm community refer to the schema theorem to explain the search strategy of genetic algorithms. For this reason it is also called the "Fundamental Theorem of Genetic Algorithms" [Gol89a]. The usual interpretation of the fundamental theorem claims that a genetic algorithm with binary representation makes an optimal allocation of the sample points given the information at hand. We believe that it is not worthwhile to discuss all the diffuse interpretations in detail. Instead, we will use a "deceptive" problem to show in this specific case what is wrong with these interpretations of the schema theorem. We will use the same deceptive problem in the next section to explain the search strategy of the PGA in simple terms.

The theory of deceptive problems has been developed by Goldberg et al. [Gol89b]. We take as example the three bit function, which is defined in the following table.

bit	value	bit	value
111	30	100	14
101	0	010	22
110	0	001	26
011	0	000	28

The function is called order-3 deceptive because we have

$$f_s(0**) > f_s(1**); f_s(*0*) > f_s(*1*); f_s**0 > f_s(**1)$$

$$f_s(00*) > f_s(11*), f_s(10*), f_s(01*)$$

$$f_s(0*0) > f_s(1*1), f_s(0*1), f_s(1*0)$$

$$f_s(*00) > f_s(*11), f_s(*10), f_s(*01)$$

and

$$\mathbf{f(000) < f(111)}$$

f_s denotes the fitness of a schema. It is defined as the average of the fitness of all points belonging to that schema, for example

$$f_s(0**) = 1/4(f(011) + f(010) + f(001) + f(000))$$

$$f_s(0**) = 19 > f_s(1**) = 11$$

The above three-bit function itself is too small to demonstrate any search strategy. Therefore Goldberg has proposed more difficult functions, which are constructed from the above three-bit function. The function **E10** consists of ten three-bit functions, where the three bits are grouped together. In the "ugly" function **U10** the three-bit subfunctions reside far apart at locations i,$i+10$,$i+20$ for $i = 1$ to 10. In both functions the fitness is defined as the sum of the subfunctions.

Functions E10 and U10 are defined on 2^{30} points. The schema $(1*..*)$ is defined on 2^{29} points. The exact fitness of the schema is given by the average of the 2^{29} function values. In a real genetic algorithm we have a much smaller population size, say 2^{12} points. Let us assume that half of the points will be members of our schema. The small number of 2^{11} points will be used by the genetic algorithm to estimate the fitness of the above schema. Holland [Hol89] correctly uses the notion of $f_s(t)$ to show that the fitness of the schema s depends on the population at generation t.

This observation leads to our main objection to the mainstream interpretation of the schema theorem.

The estimate of the fitness of a schema is equal to the exact fitness in very simple applications only. Therefore interpretations using the exact fitness cannot be applied in connection with the schema theorem. But if the estimated fitness is used in the interpretation, then the schema theorem is almost a tautology, only describing proportional selection.

This objection is similar in spirit to that of Grefenstette et al.[GB89].

Without any additional comment, we just state our second major objection against using the schema theorem to explain the search strategy of genetic algorithms. The schema theorem estimates only the disruption, i.e. the possibility of losing good substrings. The question of why the genetic algorithm builds better and better substrings by crossing-over is ignored.

The search strategy of a genetic algorithm and the PGA especially can be explained much more easily. The introduction of schemata only creates artificial confusion.

The search strategy is mainly driven by the crossover operator. It can best be described as a *scatter search* [Glo77], where new points are sampled between probabilistically chosen parent points. Because of the selection operator, the points get closer and closer. The closer the parent points are to each other, the smaller is the sampling area for new points. Furthermore, substrings which are the same in the two parent points will be transfered to the offspring. Selection and crossing-over together lead to an automatic concentration of the individuals into a promising area.

We will investigate the main components of a PGA - crossing-over and hill-climbing - with the deceptive problems E10,U10,E20 and U20. First we will make an analysis of very simple search methods. Later we will compare the results with the parallel genetic algorithm.

6 Statistical analysis of the deceptive problems

One of the simplest random search methods is *multistart hill-climbing*. An initial configuration is randomly generated and then a hill-climbing algorithm is applied. A more clever strategy is *iterated hill-climbing*. Here a new configuration for hill-climbing is generated by a mutation of the given configuration. A new configuration is accepted if it is better than the old one. This strategy can be seen as a simple evolutionary algorithm in the space of all local optima. Multistart hill-climbing has been also investigated by Ackley [Ack87]. Unfortunately he called it iterated hill-climbing. But our notion is in accordance to the notion in mathematical optimization.

We will analyse these two strategies with two hill-climbing strategies - *next ascent nahc* and *steepest ascent sahc*. In next ascent hill-climbing, the first bit of a sequence of bits is flipped if it gives an improvement. In steepest ascent hill-climbing, the bit giving the best improvement is flipped. In both cases, the iteration is stopped if no improvement can be obtained. The best-ascent hill-climbing strategy tests more configurations than the next ascent strategy. But in our ugly deceptive problem only one of the three two bit configurations leads to the optimal substring (111) with next ascent, whereas all three two bit configurations lead to the optimal string with steepest ascent. The region of attraction is larger for steepest ascent than for next ascent.

The statistical analysis of multistart hill-climbing is very simple. We define

$$prob(success) = \frac{hill.conf}{total.conf}$$

Hill.conf denotes the number of configurations leading to the global optimum if the hill-climbing strategy is applied. For an ugly deceptive problem with $n = 3m$ we get

$$hill.conf(nahc) = 2^m$$
$$hill.conf(sahc) = 2^{2m}$$

The number of tested configurations is $3m$ for nahc and $3m(m+1)$ for sahc in the worst case. Therefore we get the following result.

Theorem: *Multistart with steepest ascent hill-climbing finds the optimum with probability* 2^{-m}. *The amount of computation to find the optimum with probability almost one is* $3m(m+1) * 2^m$. *Multistart with next ascent hill-climbing needs* $3m * 2^{2m}$ *steps.*

A good hill-climbing strategy pays off the larger the problem is. We will later show that the same observation is true for the PGA.

The statistical analysis of stochastic iterated hill-climbing is more complex. In order to show the basic idea of the analysis, we start with a simple non-deceptive problem. We make the assumption that the optimum can be reached by a sequence of one bit flips. Let p be the probability of mutating one bit of the n-bit string. Let k be the current number of bits wrong. The probability of getting nearer to the optimum is the product of the probability of not flipping one of the right $n - k$ bits and the probability of flipping at least one of the k wrong bits. Ignoring higher order terms we obtain approximately

$$prob_n(k, better) = (1-p)^{n-k} * (1 - (1-p)^k) \tag{4}$$

The same analysis can also be done for the deceptive problems. We will first analyse steepest ascent hill-climbing.

We start our analysis with the observation, that after hill-climbing the string will only consists of the substrings of the two local minima (000) and (111). Two bits at least have to be flipped together to jump from (000) to (111) and vice versa. One bit flips in a substring will not lead to any changes. The probability of at least two bits flipped in three trials is $x = 3p^2(1-p) + p^3$. The two bit flips have to occur within a substring.

In the general case, let x be the probability of jumping into the attraction region of the local minima (111) from (000), given the hill-climbing method. Let y be the probability of jumping into the attraction area of (000) from (111). Ignoring higher order terms we approximately obtain for $l = 1, 2, ...$ substrings wrong

$$prob_m(l, better) = (1-y)^{(m-l)}(1 - (1-x)^l) \tag{5}$$

For next ascent hill-climbing we have $x = p^2$ and $y = 2p - p^2$. The above probability is independent of the position of the substring in the global string. In iterative hill-climbing there is no difference between a deceptive problem and an ugly deceptive problem.

Strategy	p	ET	sd	ET(6)	Conf.	sd	Nor.Conf
it-nahc-U10	0.1	1250	663	1300	37480	20010	4875
it-sahc-U10	0.1	98	54	104	11144	5989	1203
it-sahc-U10	0.18	47	24	55	7932	3995	835
it-nahc-U20	0.04	7504	3365	7703	450340	201900	51754
it-sahc-U20	0.04	684	347	681	136325	68542	14259
it-sahc-U20	0.12	144	71	152	64374	31294	6567
it-sahc-U20	0.3	953	474	1052	525267	286433	53103

Table 1: Iterated hill-climbing (128 runs each)

$ET_m(l) = 1/prob_m(l)$ is the expected number of trials needed to get at least one additional subfunction correct. In *iterated hill-climbing* we use the simple search strategy of accepting a new configuration only if it is better than the old configuration. Then the total number of trials ET(m) to reach the optimum can be computed as

$$ET(m) = ET_m(1) + ET_m(2) + .. + ET_m(initial) \tag{6}$$

Initial denotes the number of suboptimal subfunctions after the first hill-climbing. The worst case is $initial = m$. ET(m) depends on the mutation rate p. In order to minimize ET(m) all the values $ET_m(l)$ should be minimized. The optimal mutation rate for a given l can easily be computed from (5). However, this procedure is unfair, because the number of wrong substrings has to be known. The largest number of trials is always needed for the last step $ET_m(1)$. Therefore a good estimate of the mutation rate is the optimal mutation rate for $ET_m(1)$. This can easily be computed. We get for steepest ascent hill-climbing

$$x = \frac{1}{m}$$

For the mutation rate p we obtain approximately

$$p = \sqrt{\frac{1}{3m}} \tag{7}$$

In table 1 numerical experiments with different mutation rates are summarized for $m = 10$ and $m = 20$.

Conf denotes the total number of configurations evaluated, Nor.Conf are normalized configuration evaluations. These are derived as follows: During hill-climbing we need only a small number of table look-ups to evaluate a configuration, whereas m table look-ups are normally needed. In Nor.Conf we count m hill-climbing configuration evaluations as one normal configuration.

ET denotes the average number of trials to reach the optimum. It agrees quite well with the number computed by formula (6). The results confirm our statistical analysis. A mutation rate too high or too low gives an increase of the number of trials. Formula (7) gives a good estimate for the optimal mutation rate.

Furthermore, a good hill-climbing strategy pays off. At equal mutation rates, steepest ascent hill-climbing needs only one fourth of the configuration evaluations of next ascent hill-climbing.

Our analysis can easily be extended to higher order deceptive problems like Whitley's order-4 deceptive problem [WS90]. We will not do this here, but turn to the PGA.

7 The PGA and the deceptive problems

We will explain the search strategy of the PGA and its major components by solving the easy deceptive problems E10 and E20 and the "ugly" deceptive problems U10 and U20. The following features will be investigated:

- no hill-climbing
- next ascent hill-climbing
- steepest ascent hill-climbing
- 2-point cyclic crossing-over
- uniform crossing-over

Our 2-point cyclic crossing-over is implemented as follows. First the starting point of the crossing-over section is randomly drawn. Then the crossing-over interval is drawn (between 20% and 60% of the length of the string). In uniform crossing-over, each bit of the parent strings is chosen with probability 0.5. Table 2 shows the results of our numerical experiments. The mutation rates are $p = 0.1$ for $m = 10$ and $p = 0.04$ for $m = 20$.

The table can be used for many individual comparisons. We summarize the major results:

- A PGA with hill-climbing performs much better than without(wohc) hill-climbing, especially if normalized configurations are considered
- The performance of 2-point crossing-over depends on the coding. The dependency is not so strong with steepest ascent hill-climbing.
- The performance of uniform crossing-over is independent of the coding.
- A good hill-climbing strategy pays off

These results are in agreement with our results on difficult combinatorial optimization problems [Müh89] and function optimization [MSB91].

We also make a short comparison with other simulations of deceptive problems. In our PGA simulations we used a very small population size (16 or 32) in order to reduce the amount of computation. Goldberg [Gol89b] investigated the deceptive problems E10 and U10 with a standard GA of population size 2000. He reported 40000 function evaluations for the simple GA and 40600 for his messy genetic algorithm mGA, both for the function E10. The simple GA could not solve the ugly function U10, because it was run with a mutation rate of 0.

Strategy	Pop	Gen.	sd	Conf.	sd	Nor.Conf
2pc-wohc-E10	32	667	62	21398	1970	
2pc-wohc-U10	32	9979	3528	319379	112918	
2pc-nahc-E10	16	83	18	40500	8600	5245
2pc-nahc-U10	16	346	53	167400	25900	21600
2pc-sahc-E10	16	11	2	27273	4086	2890
2pc-sahc-U10	16	11	3	36066	8431	3771
2pc-nahc-E20	32	168	45	325632	85900	34990
2pc-sahc-E20	32	23	4	198678	24093	20558
2pc-sahc-U20	32	31	5	450138	66690	45010
uc–wohc-E10	32	2957	739	94669	23649	
uc–wohc-U10	32	2788	567	89260	18135	
uc–nahc-E10	16	148	27	71150	13200	7420
uc–nahc-U10	16	151	37	73152	17700	7480
uc–sahc-E10	16	7	4	22830	9804	2380
uc–sahc-E20	32	437	41	842112	86200	90490
uc–sahc-E20	32	22	5	272292	32976	27850

Table 2: PGA results (10 runs each)

Whitley [WS90] made investigations with his GENITOR II, also with a population size of 2000. He did not report the number of tested configurations, so no detailed comparisons can be made to our work. Moreover the non-distributed GENITOR was run with a mutation rate of 0. In this case it was not able to solve the "ugly" deceptive problem U10. In the distributed run an adaptive mutation rate was used. The algorithm now solved the problem. We conjecture that the main reason for this success is the mutation, not the population structure.

Our statistical analysis has shown the importance of mutation for the deceptive problems. It is not useful to compare GA runs without mutation to runs with mutation! *Mutation is a very important component of a genetic algorithm. It is a common mistake to believe that mutation is not important because the mutation rate is so small.*

8 Iterated hill-climbing vs. the PGA

The three analysed search strategies can ce described as follows. Multistart hill-climbing only generates new initial configurations for hill-climbing. There is no linkage between the searches. Iterated hill-climbing generates a new configuration in the neighborhood of the old one. If by chance an improvement is obtained, the new configuration is accepted. Iterated hill-climbing can be considered as a more restricted form of multistart hill-climbing. In the PGA new configurations are mainly generated by crossing-over. In this search strategy new configurations are built out of the components of the population of existing configurations. The recombination of the components is done randomly. How do these search strategies compare?

With the deceptive problems, multistart hill-climbing performs much worse than the other two. But iterated hill-climbing needs less computation than our PGA if a good mutaion rate is used. Why using a parallel genetic algorithm at all? The answer is parallelism.

Let us discuss the results of the experiments it-nahc-U10 and it-sahc-U20 with the PGA experiments uc-nahc-U10 and uc-sahc-U20. In both cases the PGA needs about twice the amount of computation of iterated hill-climbing. But it runs on $16(U10)$ and $32(U20)$ processors in parallel. The optimum is found after a small number of generations. The speedup of the PGA compared to iterated hill-climbing is about half the number of processors. This is a reasonable speedup for a parallel search method. But we would like to mention that in the problem domain optimization of difficult functions we could report instances of a superlinear speedup [MSB91]. This does not happen here because our deceptive problems are not difficult enough. In such problems the PGA simply does the same evaluations but in parallel. At the termination of the PGA most of the strings of the population have just one or two subfunctions incorrect.

Furthermore the comparison is a little unfair. Iterated hill-climbing strongly depends on the mutation rate. The PGA is very robust with respect to the mutation rate. We always use as mutation rate just the inverse of the length of the chromosome. In contrast the computation of a reasonable mutation rate for iterated hill-climbing needs information about the problem.

Goldberg [GDK90] suggested two extensions of the deceptive problems. One extension is to vary the size of the subfunctions, the other to vary the scale. A varying scale does not affect iterated hill-climbing because of the hard selection used. A more difficult problem for iterated hill-climbing would be deceptive problems which consists of subfunctions with different sizes, maybe nine order-3 deceptive subfunctions and one order-6 deceptive function. In this case the optimal mutation rate cannot be computed.

Our analysis has shown that deceptive problems can be nicely used to explain the search strategies of the different algorithms. But the original deceptive problems are not difficult enough to serve as good benchmarks for real optimization problems. The reason is that the local minima are too regularly spread in the fitness landscape. A mutation of just two bits leads from each of the local maxima to a better one. In our opinion it is not very useful to make the deceptive functions in small steps more and more complex and then to develop algorithms which solve these problems. A better method is to investigate the search strategies with more classical difficult benchmark optimization problems. We will do this in the next sections with the traveling salesman problem.

9 The traveling salesman problem

The famous traveling salesman problem (TSP) can be easily stated.

OPT 2 (TSP) *Given are n cities. The task of the salesman is to visit all cities once so that the overall tourlength is minimal.*

This problem has been investigated in [MGSK88] ,[GS89] and [GS91] with the PGA. The genetic representation is straightforward. The gene at locus i of the chromosome codes the edge (or link) which leaves city i. With this coding, the genes are not independent from each other. Each edge may appear on the chromosome only once, otherwise the chromosome would code an invalid tour. A simple crossing-over will also give an invalid tour. This is the reason why this simple genetic representation has not been used in genetic algorithms. The researchers tried to find a more tricky representation in order to apply a simple crossover operator.

We take the opposite approach. We use a simple representation, but an intelligent crossover operator. The crossover operator for the TSP is straightforward. It inserts part of chromosome A into the corresponding location at chromosome B, so that the resulting chromosome is the most similar to A and B. A genetic repair operator then creates a valid tour.

We call our crossover operator MPX, the maximal preservative crossover operator. It preserves subtours contained in the two parents. The pseudocode is given below.

PROC crossover (receiver, donor, offspring)

Choose position $0 <= i < nodes$ and length $b_{low} <= k <= b_{up}$ randomly.
Extract the string of edges from position i to position $j = (i + k)$ MOD $nodes$ from the mate (donor). This is the crossover string.
Copy the crossover string to the offspring.
Add successively further edges until the offspring represents a valid tour.
This is done in the following way:
IF an edge from the receiver parent starting at the last city in the offspring is possible (does not violate a valid tour)
THEN add this edge from the receiver
ELSE IF an edge from the donor starting at the last city in the offspring is possible
THEN add this edge from the donor
ELSE add that city from the receiver which comes next in the string, this adds a new edge, which we will mark as an implicit mutation.

We want to recall, that in the PGA the crossover is not done in the space of all TSP configurations, but in the space of all local minima. Our local search is a simple version of the famous 2-opt heuristic developed by Lin-Kernighan [Lin65]. Two edges are exchanged randomly. If the resulting tour is shorter, the exchange is accepted. Then the next two edges are exchanged until no exchange of two edges gives a better tour.

10 Performance evaluation for the TSP

Two different performance measures at least are used for comparing heuristics. The first measure is to compare the best solution for very large problems which each of the heuristics was able to get. The second measure is to compare the quality of the solutions after a fixed amount of time. A detailed study of the performance of the PGA based on the first measure has been done in [GS91]. In this study the

Instance	t	SA	Mult2Opt	MultLk	Gen2Opt	GenLK
EUR100	60	2.59	3.23	0.0	1.15	0.0
GRO442	4100	2.60	9.29	0.27	3.02	0.19
GRO532	8600	2.77	8.34	0.37	2.99	0.17
GRO666	17000	2.19	8.67	1.18	3.45	0.36

Table 3: Relative deviation from optimal tour length (in %)

importance of the population structure ladder on the quality of the solution was shown.

The importance of the local search in the PGA has been investigated by Ulder et al. [UPvL+91]. They asked the question: Given a fixed amount of time, is it better to use a simple, but fast local search for a large number of generations or to use a sophisticated local search for a small number of generations. In their experimental setup, the genetic algorithm had to converge within a certain time limit. Convergence is achieved when all tours in the current population have the same length or the length of the best tour did not improve within five successive generations. This condition forced the population sizes of the genetic algorithm to be very small. The experiment was done on a sequential computer.

Ulder et al. compared the standard 2-opt with the more complicated Lin-Kernighan [LK73] neighborhood. In the latter case they used a pair of improvement operators, the dynamical k-swap and the additional 4-swap as described in [LK73]. The results are shown in table 3.

The problem instances starting with GRO can be found in [GH88]. GRO532 is the Padberg-Rinaldi problem mentioned above. The reference point is given by simulated annealing SA according to the cooling schedule of Aarts et al. [AK89]. The results demonstrate the advantage of the MPX crossing-over. The genetic versions Gen2Opt and GenLK of 2-Opt and Lin-Kernighan, respectively, perform clearly better than their multistart companions, despite the penalty of the convergence constraint. Moreover, GenLK is superior to the other algorithms. This shows the advantage of using a sophisticated local search method in the PGA.

The absolute quality of the above solutions is worse than the solutions we obtained with the PGA [GS91]. This can be explained by the experimental setup, which required a convergence within the time limit given by simulated annealing.

In combining these results on local search and our results on population structure in the PGA, it seems obvious that a good PGA implementation should use at least two different local search methods. Most of the population should use a simple, but fast local search. But one or more processors should perform a very good local search like the Lin-Kernighan heuristic. Each time a new best-so-far solution has been found, one of the processors with the sophisticated local search method should be initialized with that solution.

We have not yet done this implementation for the TSP, because we believe that highly sophisticated implementations of an *iterated Lin-Kernigham algorithm* will yield semi-optimal solutions faster than even our PGA. The reason is that the Lin-

Kernigham heuristic is very good for Euclidean TSP-problems. The situation is different with other combinatorial problems like e.g. the graph partitioning problem, where the Lin-Kernigham heuristic performs much worse.

In the next section we investigate by a configuration space analysis why Gen2Opt is better than Mult2Opt.

11 Configuration space analysis of the TSP

The analysis of the TSP is more difficult than that of the deceptive problems because the genetic coding is more complex. In the TSP we have $n-1$ alleles at each loci instead of two. Furthermore, the alleles are not independent. In our configuration space analysis the following questions will be investigated

- How many different edges take part in 2-opt local minima?
- How big is the overlap between two 2-opt local minima
- Which edges are most likely contained in 2-opt local minima?

It is impossible to answer these questions by analytical methods. The configuration space analysis can only be done by computation. We have chosen TSP problems of order 40,60,80 and 100 as problem instances. The cities have been placed randomly. It is outside the scope of this paper to discuss whether TSP problems with random drawn cities are representative for the class of all TSP problems. We have to mention that our conjectures are not valid for degenerate TSP instances like cities arranged on a grid or on a circle. On a circle, there exists only one 2-opt tour. On a grid, the 2-opt tours have a special structure.

A similar configuration space analysis has been done by Kirkpatrick [KT85]. He investigated a different TSP problem, where all the distances are drawn randomly. This is a non geometric TSP problem.

Our analysis is based on 800 different 2-opt tours, which have been generated from different initial tours. Table 4 gives the number of different edges and the average distance between two tours. The difference is defined as the Hamming distance i.e the number of different edges. AHD denotes the average distance, HD10 the average distance of the best 10 tours, HD50 the average distance of the best 50 tours.

n	no edges	AHD	HD10	HD50
40	140	13.2	7.1	9.0
60	234	20.0	11.4	13.9
80	340	28.0	17.5	21.4
100	417	38.0	24.8	30.2

Table 4: Different edges and average distance in 2-opt solutions

The data of the table suggest a conjecture.

Conjecture 1 *The number of edges of 2-opt local minima of random TSP problems is bound approximately by* $4.3 * n$ *The average distance between two 2-opt tours is approximately* $1/3 * n$.

We are well aware, that the above conjectures can only be proved by exhaustive search. But we have so much evidence, that we firmly belive that the conjecture is correct. Kirkpatrick [KT85] conjectured a bound of $3 * n$ for his TSP problem and an average distance of $1/3 * n$. In addition, our table shows that the better the tours are, the more similar they are.

The bound on the number of edges implies the following. If we combine all 2-opt tours into a single supergraph, then on the average $2 * 4.3 = 8.6$ edges are only connected to a given city. This result is interesting in itself and deserves further study.

Our main question is, why the MPX crossover is so successful on TSP problems. The answer is surprisingly simple. It is based on the following observation.

Conjecture 2 *The edges of the best 2-opt tours of random TSP problems are most likely contained in other 2-opt tours also.*

In order to explain this conjecture we define the *frequency of occurrence* of the edges of a tour. It is defined as follows. For each of the given 2–opt tours we compute how often its edges are in other 2-opt tours also. We sum these numbers and divide the sum by the product of the total number of tours and the number of cities. The frequency of occurrence is $1/n$, if the edges of all tours are different , and 1, if all tours are equal. The following table shows the result

n	aver	max	min	r
40	0.62	0.69	0.53	-0.78
60	0.67	0.73	0.56	-0.78
80	0.64	0.70	0.56	-0.85
100	0.62	0.67	0.55	-0.79

Table 5: Frequency and correlation r of pathlength to frequency

The average frequency of occurence is approximately $2/3n$. This result can be interpreted as each edge being contained on the average in 2/3 of the other 2-opt tours. The correlation coefficient r tests the hypothesis of a correlation between the pathlength and the frequency. The computed values indicate that tours with small pathlength have a higher frequency of occurence in the pool.

This fact is also shown in the next table. Here the pool of edges is restricted to the best 10 2-opt tours, the best 100 2-opt tours etc.

N	aver100	aver 80	aver60	aver40
10	0.78	0.80	0.83	0.84
100	0.69	0.72	0.74	0.74
200	0.67	0.70	0.72	0.70
400	0.65	0.68	0.70	0.67
800	0.62	0.64	0.67	0.62

Table 6: Frequency of occurrence for the best N tours

The table shows that the better the solutions, the more similar they are. This observation does not imply, that crossing-over of just two strings is the best way

to do the linkage. We have shown in [Müh89] that a voting recombination of seven parents works very well in the quadratic assignment problem. A comparison of voting recombination of several TSP tours vs. crossing-over of two tours is a topic of future research.

We summarize the result of this analysis in two statements.

The combined power of selection and crossover *Better 2-opt tours are more similar. By combining two good tours, the probability of obtaining a still better tour is higher than by combining two arbitrary 2-opt tours.*

The ideal PGA *Combine the genetic material of two individuals to produce better individuals. By selection of the better individuals decrease the genetic variety of the population, so that the chance of producing a better individual remains high. But take care that the genetic material of the best individual remains in the gene pool*

Unfortunately, the PGA does not know the best individual beforehand. If selection eliminates some of the genetic material of the best individual, it will converge to suboptimal solutions.

12 Conclusion

This paper has described the *clean* parallel genetic algorithm PGA, clean in the sense that it relies on a single strategy only. The clean algorithm has been successfully applied in a number of difficult optimization problems. In the paper we have shown the importance of a population structure, a good hill-climbing strategy and a problem dependent crossover operator. We have mentioned some extensions which will make the PGA still more effective. The most important ones are

- some individuals do a different local search
- the population structure may change during the run

A computational optimal PGA would start with very simple and fast local search methods. The search space is broadly explored mainly by crossing-over. In the middle of the search, the PGA should also use individuals with very good search methods. At the end of the algorithm, similar individuals should die out.

It is very easy to extend the PGA to real life optimization problems with conflicting goals. Here a group of species can be used for solving problems in cooperation or in competition.

The PGA is also a contribution to evolutionary theory. It has shown the importance of a spatial population structure for the evolution of a species in a rough fitness landscape. The success of the PGA suggest exploring other problem solving metaphors also. Why not use the market economy as a parallel search metaphor? A comparison of problem solving by a market framework and by biological evolution on the same set of artificial problems would give further insight into economy and biology.

References

[Ack87] D. Ackley. *A Connectionist Machine for Genetic Hillclimbing.* Kluver Academic Publisher, Boston, 1987.

[AK89] E. Aarts and J. Korts. *Simulated Annealing and Boltzmann Machines.* John Wiley, New York, 1989.

[CHMR87] J.P. Cohoon, S.U. Hedge, W.N. Martin, and D. Richards. Punctuated equilibria: A parallel genetic algorithm. In J.J. Grefenstette, editor, *Proceedings of the Second International Conference on Genetic Algorithms*, pages 148–154. Lawrence Erlbaum, 1987.

[Fel75] J. Felsenstein. A pain in the torus: Some difficulties with models of isolation by distance. *Amer. Natur.*, 109:359–368, 1975.

[Fel76] J. Felsenstein. The theoretical population genetics of variable selection and migration. *Ann. Rev. Genet.*, 10:253–280, 1976.

[Fis58] R. A. Fisher. *The Genetical Theory of Natural Selection.* Dover, New York, 1958.

[GB89] J.J. Grefenstette and J.E. Baker. How genetic algorithms work: A critical look at implicit parallelism. In H. Schaffer, editor, *3rd Int. Conf. on Genetic Algorithms*, pages 20–27. Morgan-Kaufmann, 1989.

[GDK90] D.E. Goldberg, K. Deb, and B. Korb. Messy genetic algorithms revisited: Studies in mixed size and scale. *Complex Systems*, 4:415–444, 1990.

[GH88] M. Grötschel and O. Holland. Solution of large-scale symmetric traveling salesman problems. Technical report, Institut f. Ökonometrie und Operations Research, University of Bonn, Report No. 88506-OR, 1988.

[Glo77] F. Glover. Heuristics for integer programming using surrogate constraints. *Decision Sciences*, 8:156–166, 1977.

[Gol89a] D.E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison-Wesley, Reading, 1989.

[Gol89b] D.E. Goldberg. Messy genetic algorithms: Motivation, analysis, and first results. *Complex Systems*, 3:493–530, 1989.

[GS89] M. Gorges-Schleuter. Asparagos: An asynchronous parallel genetic optimization strategy. In H. Schaffer, editor, *3rd Int. Conf. on Genetic Algorithms*, pages 422–427. Morgan-Kaufmann, 1989.

[GS91] M. Gorges-Schleuter. *Genetic Algorithms and Population Structures - A Massively Parallel Algorithm.* PhD thesis, University of Dortmund, 1991.

[Hol75] J.H. Holland. *Adaptation in Natural and Artificial Systems.* Univ. of Michigan Press, Ann Arbor, 1975.

[Hol89] J.H. Holland. Searching nonlinear functions for high values. *Appl. Math. and Comp.*, 32:255–274, 1989.

[KT85] S. Kirkpatrick and G. Toulouse. Configuration space analysis of travelling salesman problems. *J.Physique*, 46:1277–1292, 1985.

[Lin65] S. Lin. Computer solutions of the traveling salesman problem. *Bell. Syst. Techn. Journ.*, 44:2245–2269, 1965.

[LK73] S. Lin and B. W. Kernighan. An efficient heuristic for the traveling salesman problem. *Operations Research*, 21:298–516, 1973.

[MGSK87] H. Mühlenbein, M. Gorges-Schleuter, and O. Krämer. New solutions to the mapping problem of parallel systems - the evolution approach. *Parallel Computing*, 6:269–279, 1987.

[MGSK88] H. Mühlenbein, M. Gorges-Schleuter, and O. Krämer. Evolution algorithms in combinatorial optimization. *Parallel Computing*, 7:65–88, 1988.

[MS89] B. Manderick and P. Spiessens. Fine-grained parallel genetic algorithm. In H. Schaffer, editor, *3rd Int. Conf. on Genetic Algorithms*, pages 428–433. Morgan-Kaufmann, 1989.

[MSB91] H. Mühlenbein, M. Schomisch, and J. Born. The parallel genetic algorithm as function optimizer. *Parallel Computing*, 17:619–632, 1991.

[Müh89] H. Mühlenbein. Parallel genetic algorithm, population dynamics and combinatorial optimization. In H. Schaffer, editor, *3rd Int. Conf. on Genetic Algorithms*, pages 416–421, San Mateo, 1989. Morgan Kaufmann.

[PS89] C. Peterson and B. Söderberg. A new method for mapping optimization problems onto neural networks. *Int. J. Neural Syst.*, 1:995–1019, 1989.

[Rec73] I. Rechenberg. *Evolutionsstrategie - Optimierung technischer Systeme nach Prinzipien der biologischen Information.* Fromman Verlag, Freiburg, 1973.

[Sch81] H.-P. Schwefel. *Numerical Optimization of Computer Models.* Wiley, Chichester, 1981.

[Sch89] H. Schaffer, editor. *Procedings of the Third International Conference on Genetic Algorithms*, San Mateo, 1989. Morgan Kaufmann.

[Tan89] R. Tanese. Distributed genetic algorithm. In H. Schaffer, editor, *3rd Int. Conf. on Genetic Algorithms*, pages 434–440. Morgan-Kaufmann, 1989.

[UPvL+91] N.L.J. Ulder, E. Pesch, P.J.M. van Laarhoven, H.-J. Bandelt, and E.H.L. Aarts. Improving tsp exchange heuristics by population genetics. In R. Maenner and H.-P. Schwefel, editors, *Parallel Problem Solving from Nature*, pages 109–116. Springer-Verlag, 1991.

[vLM91] G. von Laszewski and H. Mühlenbein. A parallel genetic algorithm for the graph partitioning problem. In R. Maenner and H.-P. Schwefel, editors, *Parallel Problem Solving from Nature*, Lecture Notes in Computer Science 496, pages 165–169. Springer-Verlag, 1991.

[Wri32] S. Wright. The roles of mutation, inbreeding, crossbreeding and selection in evolution. In *Proc. 6th Int. Congr. on Genetics*, pages 356–366, 1932.

[WS90] D. Whitley and T. Starkweather. Genitor ii: a distributed genetic algorithm. *J. Expt. Theor. Artif. Intell.*, 2:189–214, 1990.