Locality in Genetic Algorithms

V. Scott Gordon†

Abstract — **This paper attempts to quantify spatial locality in various genetic algorithms. In particular, the following algorithms are examined: SGA (Goldberg's standard genetic algorithm), several "island" models, and two cellular algorithms (fixed topology and random walk). The approaches are also applicable to Evolution Strategies that employ methods such as recombination or parameter averaging. Two different locality metrics are presented: percentage of remote references (for parallel machines with a few processors), and traffic per link (for massively parallel machines). We derive expressions for computing locality in this manner, and discuss the utility, implications, and limitations of our results.**

I. Introduction

In order to better characterize the exploitable parallelism in genetic algorithms (GA), it is useful to examine their inherent locality. Locality issues, important in any parallel computing application, have been addressed only indirectly by genetic algorithm researchers. Genetic algorithms are, possibly, an ideal setting for locality analysis because they consist primarily of repeated operations on one large data structure (i.e., the population of strings). The development of a unified approach to measuring locality in genetic algorithms would significantly improve understanding of appropriate parallel GA models. There currently are no standard methods for measuring locality, so various approaches will need to be considered. Verification of the accuracy of any particular metric may be difficult. At this time there exists no single measure of locality for genetic algorithms because each one targets machine characteristics which are not necessarily comparable. The approaches presented in this paper for genetic algorithms are also applicable to Evolution Strategies that employ methods such as recombination or parameter averaging.

II. Locality – background

There are two types of locality. *Temporal locality* indicates that if a data element is referenced, it is likely to be referenced again soon. This paper concentrates on *Spatial locality*, which refers to the likelihood that if a data element is referenced, a nearby element will also likely be referenced. In a parallel machine, spatial locality issues coincide with the memory hierarchy configuration. That is, data elements are stored either in local memory (possibly cache) or elsewhere (remote or global) depending on locality considerations.

Locality differs between the genetic algorithms, notably with respect to parent selection. Standard genetic algorithms implement selection such that each recombination may involve *any* two strings from the population. On the other hand, selection in an Island model restricts recombination to individuals within a subpopulation, and therefore has more spatial locality. Further, if the Island model were implemented such that each sub-population resides on a separate processor, the only remote references would be those that involve the periodic migration of individuals between subpopulations. In this case, spatial locality is largely affected by the migration rate.

Analysis of spatial locality implies a concept of "distance" between data elements. In genetic algorithms, this can be expressed in various ways depending on how the algorithm is mapped onto a parallel machine. For example, if subpopulations are distributed on a hypercube, one per node, then the distance between two subpopulations could be expressed as the number of links that must be traversed to get from one to the other (the upper bound is $\log(d)$ where d is the number of nodes in the hypercube). Sometimes distance is expressed only as "local" or "remote".

The effect of locality on performance relates to the bandwidth and latency of the communication mechanism for the host machine, as well as the nature of the algorithm [4]. Algorithms with high locality can more easily utilize distributed memory systems than can algorithms with low locality. Sometimes the ideal mapping (for locality) of the algorithm may not match perfectly with the available hardware. For example, it may be necessary to implement an Island model with 8 subpopulations on a multiprocessor system with 4 processors, because of hardware constraints.

III. SGA

Golberg's simple genetic algorithm, generally referred to as *SGA* [2], can be parallelized across $p/2$ processors (where p is the population size) by utilizing *tournament selection* to select parents [3]. Each processor extracts t individuals at random from a global population (t is the *tournament size*), and then selects two of the t individuals for reproduction. Typically, selection of the two is biased towards better individuals. The resulting offspring occupy two particular locations in the subsequent generation.

We shall assume that each processor has it's own local memory, and that the population is divided up evenly amongst the processors. If we also assume that $p/2 > n$, where n is the number of available processors, then the number of "iteration pairs" i_p per processor (the number of pairs of individuals each processor is responsible for generating at each generation) is:

$$i_p = \frac{p}{2n} \tag{1}$$

Each iteration-pair must fetch t elements from the population, and put 2 elements back in. The probability that a particular data access is remote is $(n-1)/n$, and the probability that a particular data access is local is $1/n$. Thus, for

† Dept of Computer Science, Colorado State Univ.
email: gordons@cs.colostate.edu

one iteration-pair, the expected number of remote accesses per generation is:

$$E(remote) = \frac{t(n-1)}{n} \tag{2}$$

and the expected number of local accesses (including the two "put" accesses described earlier) per iteration-pair per generation is:

$$E(local) = \frac{t}{n} + 2 \tag{3}$$

Since all i_p iteration pairs must access the same global population, it is logical to express spatial locality in terms of the *percentage of remote references*, where a smaller percentage implies a higher locality. Since each iteration-pair is processed independently, $E(remote)/E(local)$, which is defined for iteration-pairs, is equally valid for the entire population. Thus we could examine the locality of SGA by examining the ratio of local versus remote references:

$$\frac{remote}{local} = \frac{\frac{t(n-1)}{n}}{\frac{t}{n}+2} = \frac{t(n-1)}{t+2n} \tag{4}$$

for various values of t, n, and p. It follows that the percentage of remote accesses is:

$$\%remote_{sga} = \frac{t(n-1)}{(t+2n)+t(n-1)} = \frac{t(n-1)}{n(t+2)} \tag{5}$$

For fixed t, clearly the percentage of remote accesses depends solely on the number of processors. For $t = 4$, this approaches 66% as shown in Figure 1:

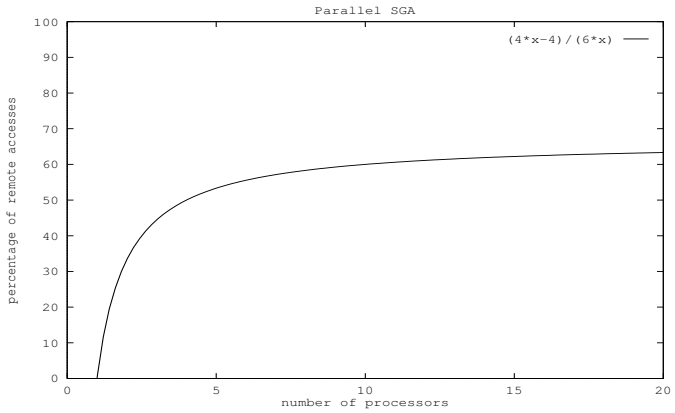

Figure 1: Locality of SGA

The impact of such a high percentage of remote accesses also depends on the *total* number of accesses, which in turn depends on the population size. The total number of accesses per iteration-pair per generation is $t + 2$ (t accesses for the selection tournament and 2 for replacement), and thus the total number of accesses is $p(t+2)/2$ (there are $p/2$ iteration pairs per population). Assuming a tournament size of four, the total accesses per generation equals $6p/2 = 3p$.

IV. Island Models

In an island model, several single population genetic algorithms are run separately, each with their own subpopulation [5]. Information is shared between islands by allowing strings to migrate between islands at predefined intervals. Migration is characterized by *migration topology*, *migration rate*, and *number of strings* involved in the migration. For purposes of this analysis we will assume that remote references (i.e., those that access data in another island) all involve the same amount of overhead. Thus the migration topology is irrelevant, and we need only consider migration rate and number of migrating strings.

We assume that the population of strings is spread uniformly across the islands, and that each island is contained on a separate processor. Initially we assume that each island implements SGA. Later we will describe how the results can be adjusted for other algorithms.

The migration rate is defined here as the number of generations between migrations, also called the *swap interval*. Let s equal the swap interval. During s generations, each subpopulation performs a total of $s \cdot d$ local data accesses, where d is the number of accesses which occur in a single generation of SGA. In one generation of SGA (where SGA is as defined earlier), there are $p_s/2$ iteration pairs, each of which performs $t + 2$ local accesses, where p_s is the subpopulation size and t is the size of the selection tournament. If n is the number of subpopulation, then the total number of local accesses which occur in a single swapinterval s is:

$$d = \frac{n \cdot s \cdot p_s \cdot (t+2)}{2} \tag{6}$$

After s generations, migration takes place. This requires m remote access per subpopulation, where m is the number of strings migrated. Since there are n subpopulations performing the migration, there are $m \cdot n$ total remote references. The percentage of remote references is therefore:

$$\%remote_{island} = \frac{mn}{\frac{p_s sn(t+2)}{2} + mn} = \frac{2m}{p_s s(t+2) + 2m} \tag{7}$$

Assuming a tournament size of 4:

$$\%remote_{island} = \frac{m}{4 \cdot s \cdot p_s + m} \tag{8}$$

We can easily tabulate a variety of values. If we consider swap intervals of 5 and 50, subpopulation sizes of 20, 100, and 1000, and string migrations of size of 1 and 5, we get the values shown in Table 1.

Adjusting for other models

Adjusting equations 6, 7, and 8 for other models (i.e., replacing each island of SGA with an island of Genitor or some other genetic algorithm) is straightforward.

For Genitor islands, replace equation 6 with:

$$d = n \cdot s \cdot p_s \cdot (t+1) \tag{9}$$

since each genitor recombination produces *one* offspring rather than two, and thus the notion of iteration-pairs no longer exists. Equations 7 and 8 are adjusted similarly.

For parallel-CHC islands (described by Gordon *et al.* [3]), replace equation 6 with:

$$d = \frac{n \cdot s \cdot p_s \cdot (t+4)}{2} \tag{10}$$

% of remote refs for Island-SGA			
parameter values			%remote
S = 5	M = 1	Ps = 20	0.2499%
"	"	Ps = 100	0.0499%
"	"	Ps = 1000	0.0049%
"	M = 5	Ps = 20	1.2345%
"	"	Ps = 100	0.2494%
"	"	Ps = 1000	0.0249%
S = 50	M = 1	Ps = 20	0.0249%
"	"	Ps = 100	0.0049%
"	"	Ps = 1000	0.0005%
"	M = 5	Ps = 20	0.1248%
"	"	Ps = 100	0.0249%
"	"	Ps = 1000	0.0025%

Table 1: Locality in island models. S = swap interval, M = number of strings migrated, P_s = subpopulation size.

since each iteration-pair must additionally access two particular elements to determine whether replacement is to occur or not. Equations 7 and 8 are adjusted similarly.

Finally, for cellular islands, replace equation 6 with:

$$d = n \cdot s \cdot p_s \cdot demesize \tag{11}$$

where *demesize* is the number of strings included in each neighborhood (the rest is logically similar to Genitor). Again, equations 7 and 8 are adjusted similarly.

IV. Cellular Models

Cellular genetic algorithm (CGA) models (sometimes called *massively parallel* or *fine grain* genetic algorithms) are designed for machines with a large number of processors (1000 or more). In the simplest case one can use a single large population with one string per processor. In order to avoid high communication overhead, these algorithms generally impose some limitation on mating based on distance [1].

We shall assume that strings reside on a two-dimensional grid, and that population size is a perfect square. Edge elements wrap around, forming a torus. Each individual is processed in parallel at each generation. Two approaches are considered. In the *fixed topology* method, demes (neighborhoods) consist of specific nearby individuals. In the *random walk* method, demes consist of the set of strings residing along a random walk starting from the node in question. In both cases, an offspring replaces the node in question if it has a higher fitness. Whitley showed that the fixed topology method is actually a class of cellular automata with probabilistic rewrite rules, and suggested the name "cellular genetic algorithm" [6].

There is no intrinsic requirement that each string reside on a separate processor. It is possible to divide the entire population into logically adjacent subgrids, and then distribute the subgrids across processors. It is interesting to consider both cases; i.e., where there exists one string per processor and also where there exists one subgrid per processor.

A. One String per Processor

When each string resides on a separate processor, our approach is to examine the data transmission requirements for each node during recombination. A simplified method is to count the total number of string (or fitness value) transmissions per link, T, that a given individual (i.e., process – since there is one string per processor) requests in selecting a mate. This information will give us an indication of the total communication load in the system, since every string undergoes recombination at each generation.

In a two-dimensional grid such as was described above, there are $2n$ links, where n is the number of nodes in the grid. If each node spawns T transmissions per link, it follows that the total demand of network links is nT, and the strings transmitted per link is:

$$strings/link = (n \cdot T)/(2 \cdot n) = T/2 \tag{12}$$

Fixed Topology

In a fixed topology with a deme size of 4, we assume that each string selects a mate from the strings directly above, below, to its left, and to its right. A simple method is for them to merely transmit themselves. Thus $T = 4$, and from equation (12) the number of strings per link is $T/2 = 2$.

For a deme size of 8, we assume that selection occurs among the strings in the square surrounding the node in question. In this case, there are two approaches. First, we can simply assume that each string transmits itself, and in this case four nodes (those directly adjacent) traverse a distance of one link, and four node (those on the corners of the square) traverse a distance of two links. In this case, $T = 12$, and from equation (12) the number of strings per link is 6. Alternately, nodes could just transmit their fitness values, in which case it may be possible for two of the adjacent nodes to packetize their own fitness values along with those of the corner nodes into a single transmission. In this case, the number of transmissions is 8, plus another 1.5 (on average) to retrieve the actual selected string. Thus $T = 9.5$ and from equation (12) the number of strings per link is 4.75.

Random Walk

In a *random walk* algorithm, a short random traversal of part of the grid is made, and a superior string is ultimately returned to the original node. Thus there are data transmissions associated with the walk, and additional data transmissions associated with returning the mate. Clearly the number of data transmissions associated with the walk is equal to the walk length. The number of data transmissions associated with returning the mate equals the average distance from the original node to the last node in the walk.

For a walk length of three, it is simple to count the number of ways that a walk can end up on various surrounding nodes. In this case, there are 24 paths which end up on nodes which are three links away from the original node, and 36 paths which end up on nodes which are one link away from the original node. Thus the average distance is $[3(24) + 1(36)]/60] = 1.8$. Thus $T = 3 + 1.8 = 4.8$ and from equation (12) the number of strings per link is 2.4.

B. One Subgrid per Processor

A logical approach to examining the locality of a CGA in which strings are contained in subgrids, each of which reside on separate processors, is to compare the percentage of remote references. Here the percentage of remote references increases as the the percentage of strings at the edges of the subgrids increases. This leads to the apparent anomaly that locality (and thus performance) degrades as the number of processors increase (assuming that total population size is fixed). Of course, this actually represents a trade-off since more processors also means more parallelism.

One way to minimize the percentage of edge strings is to divide the total population, which is assumed to be a square, into equal subgrids which are themselves squares. Next, we define n to be the total number of nodes in the population and n_s to be the number of nodes in a subgrid. The length of a side of the population grid is then $\sqrt{n}$, and $\sqrt{n_s}$ is the length of a side of a subgrid. Finally, we assume that all remote accesses have the same overhead.

Fixed Topology

Since we are assuming a torus, with wrap-around edge elements, each subgrid is structurally equivalent. Therefore the percentage of remote accesses is the same for a subgrid as it is for the entire population. Thus we need only calculate the percentage of remote accesses for a single subgrid.

In a subgrid, all interior nodes mate only with nodes inside the subgrid. Edge nodes may mate with nodes within the grid, or with nodes in one or more other subgrids. In particular, for a deme size of 4, the process of reproduction for one interior node results in 6 local data accesses (4 for selecting a mate, one for retrieving the contents of the node itself, and one for replacing the node with the offspring). Nodes at the corners of the subgrid require 4 local data accesses and 2 remote accesses (2 remote accesses and 2 local accesses for selecting a mate, one local access for retrieving the contents of the node itself, and one local access for replacement). Nodes at the sides of the subgrid similarly require 5 local accesses and 1 remote access. The number of corner nodes is always 4, the number of side nodes is $4 \cdot \sqrt{n_s} - 8$, and the number of interior nodes is $(\sqrt{n_s} - 2)^2$. Assuming $n_s \geq 9$, the total number of remote accesses A_r and local accesses A_l is:

$$A_r = (2 \cdot 4) + [1 \cdot (4 \cdot \sqrt{n_s} - 8)] = 4\sqrt{n_s} \quad (13)$$

$$A_l = 6n_s - 4\sqrt{n_s} \quad (14)$$

It follows that the percentage of remote accesses is:

$$\%remote_{grid-4} = \frac{4\sqrt{n_s}}{6n_s} = \frac{2}{3\sqrt{n_s}} \quad (15)$$

For a deme size of 8, the process of reproduction for one interior node results in 10 local data accesses (8 for selecting a mate, one for retrieving the contents of the node itself, and one for replacing the node with the offspring). Nodes at the corners of the subgrid require 5 local data accesses and 5 remote accesses (5 remote accesses and 3 local accesses for selecting a mate, one local access for retrieving the contents of the node itself, and one local access for replacement). Nodes at the sides of the subgrid similarly require 7 local accesses and 3 remote accesses. The numbers of each type of node the same as described earlier. Again, assuming $n_s \geq 9$, the total number of remote accesses A_r and local accesses A_l is:

$$A_r = (5 \cdot 4) + [3 \cdot (4\sqrt{n_s} - 8)] = 12\sqrt{n_s} - 4 \quad (16)$$

$$A_l = 10n_s - 12\sqrt{n_s} + 4 \quad (17)$$

It follows that the percentage of remote accesses is:

$$\%remote_{grid-8} = \frac{12\sqrt{n_s} - 4}{10n_s} = \frac{6\sqrt{n_s} - 2}{5n_s} \quad (18)$$

Figure 2 contains a plot of equations 15 and 18.

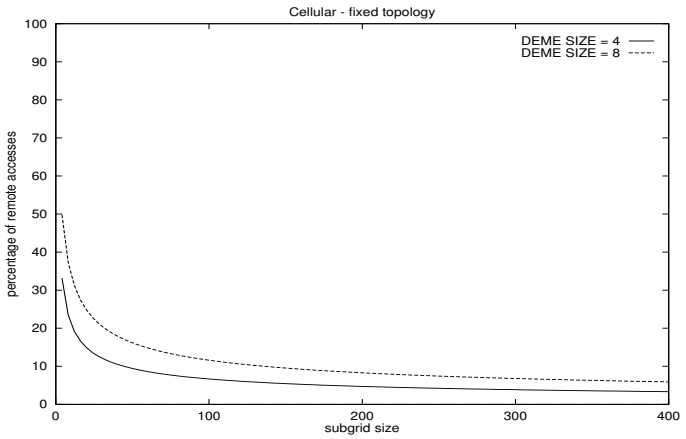

Figure 2: Locality of fixed topology CGA, various deme sizes.

Random Walk

Determining the percentage of remote references in a random walk with one subgrid per processor requires considering the various cases where a walk could leave the local subgrid. Nodes which are closest to the edges are more likely to generate remote references than nodes which are not as near to an edge. Nodes near corners may generate remote references to more than one external processor. There are many machine-dependent considerations, such as the overhead associated with obtaining one element from each of two processors versus obtaining two elements from one processor.

In order to simplify the analysis, we make the following assumptions. First, we assume that all remote accesses incur the same overhead. Thus the overhead associated with two remote references is twice that of a single reference, regardless of whether they are from the same processor or different processors. Second, we assume that the walk is generated ahead of time, and then all data accesses are made.

Figure 3 contains two charts: the *template grid* (left) contains the number of ways that neighboring cells can be traversed (i.e., how many paths traverse each neighboring cell). The *edge grid* (right) labels the cells which are close to the edges of a subgrid.

The total number of possible data references is the sum of the values in the template grid, which equals 172. By overlaying the template grid on top of each cell in the edge grid, it is possible to determine how many of these 172 possible references would be remote. Enumerating these then becomes relatively simple for each category of edge elements *a-i* and *x* in the edge grid:

			1			
		3	4	3		
	3	8	20	8	3	
1	4	20	16	20	4	1
	3	8	20	8	3	
		3	4	3		
			1			

template grid

x	x	x	i	h	g
x	x	x	i	h	g
x	x	x	i	h	g
i	i	i	f	e	d
h	h	h	e	c	b
g	g	g	d	b	a

edge grid

Figure 3: The *template grid* shows how many ways neighboring cells can be traversed during a random walk of length 3 starting from the center cell. The *edge grid* shows the categories of edge and corner cells which access remote cells. Overlaying the template grid on top of each cell in the edge grid allows one to count the fraction of remote references.

(a nodes) $Remote = 92/172$ (f nodes) $Remote = 2/172$
(b nodes) $Remote = 61/172$ (g nodes) $Remote = 53/172$
(c nodes) $Remote = 22/172$ (h nodes) $Remote = 11/172$
(d nodes) $Remote = 54/172$ (i nodes) $Remote = 1/172$
(e nodes) $Remote = 12/172$ (x nodes) $Remote = 0/172$

Note that for all interior nodes (those that are at least three cells away from an edge, marked "x" in Figure 3), the percentage of remote references is 0. Assuming that $n_s \geq 36$, the number c_t of each type of node t in any subgrid is:

$c_a = 4$ $c_d = 8$ $c_g = 4\sqrt{n_s} - 24$ $c_x = (\sqrt{n_s} - 6)^2$
$c_b = 8$ $c_e = 8$ $c_h = 4\sqrt{n_s} - 24$
$c_c = 4$ $c_f = 4$ $c_i = 4\sqrt{n_s} - 24$

It follows that the fraction of remote accesses is:

$$remote_{rw} = \left[4(\frac{116}{172}) + 8(\frac{127}{172}) + (4\sqrt{n_s} - 24)(\frac{65}{172})\right] / n_s$$
$$= \frac{65\sqrt{n_s} - 20}{43 n_s} \quad (19)$$

Equation 19 does not include the 2 local accesses necessary to fetch and to replace the particular node in question. Including this reduces the fraction of remote accesses to:

$$remote_{rw} = \frac{39\sqrt{n_s} - 12}{43 n_s} \quad (20)$$

Figure 4 shows a plot of equation 20.

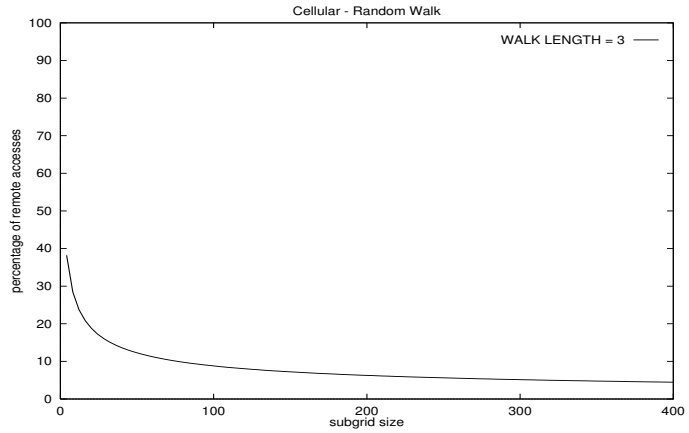

Figure 4: Locality of random walk CGA, walk length = 3.

V. Summary and Conclusions

Our analyses included two different approaches to measuring locality: percentage of remote references (for parallel machines with a few processors), and traffic per link (for massively parallel machines). Most of the analysis is for parallel machines with a few processors, so we can tabulate this data for several scenarios. Table 2 contains locality calculations for one generation using the expressions described earlier, for a 4-processor machine. For the Island model, swapinterval is 5 and number of strings migrated is 5. Quantifying locality effects in this manner illustrates which algorithms are more appropriate for particular types of parallel hardware.

Various GAs. 4 processors, popsize of 400, 2500			
	Refs	# Rem	% Rem
SGA, $p = 400$	1200	600.0	50.00
SGA, $p = 2500$	7500	3750.0	50.00
Isl-SGA, $p = 4$x(100)	1204	2.4	0.20
Isl-SGA, $p = 4$x(625)	7504	3.0	0.04
Grid4, $p = 4$x(10x10)	2400	160.0	6.67
Grid4, $p = 4$x(25x25)	15000	400.0	2.67
Grid8, $p = 4$x(10x10)	4000	464.0	11.60
Grid8, $p = 4$x(25x25)	25000	1184.0	4.74
RWalk3, $p = 4$x(10x10)	2000	176.4	8.79
RWalk3, $p = 4$x(25x25)	12500	448.0	3.58

Table 2: Summary of locality for some genetic algorithms

There are several factors which have not been considered in these analyses. For example, one might expect that islands divided up into subgrids (in the manner described in Section 4.b) would perform slower than a straight-forward island model because the subgrids are required to run synchronously (there are remote accesses at each generation). Thus in the one-subgrid-per-processor model, each generation can only run as fast as the speed of a distant access. Naturally, machine-dependent factors also play a big role in the performance of any parallel algorithm application.

Various algorithms will differ in their problem-solving characteristics, and balancing potentially competing factors (e.g., problem-solving power, parallelism, overhead) is facilitated by numerical methods. The ability to measure locality may ultimately allow us to compute tradeoffs more exactly. Until now, locality considerations have not been addressed rigorously by the GA community. Hopefully, our first steps will lead to more complete and more universal metrics.

References

[1] Y. Davidor. A Naturally Occurring Niche & Species Phenomenon: The Model and First Results. *Proc. of the 4th ICGA*, Morgan-Kaufmann, 1991

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

[3] V. Gordon, D. Whitley, and A. Böhm. Dataflow Parallelism in Genetic Algorithms. *PPSN2*, North Holland, 1992

[4] A.Z. Spector. Achieving Application Requirements, in *Distributed Systems*. Ed. S. Mullender, ACM Press, ©1989

[5] D. Whitley and T. Starkweather. Genitor II: a Distributed Genetic Algorithm. *J. Exp. Theor. Artif. Intell.*, 1990

[6] D. Whitley. Cellular Genetic Algorithms. *Proc. of the 5th ICGA*, Morgan-Kaufmann, 1993