Comparison of Parallel Messy Genetic Algorithm Data Distribution Strategies[*]

Laurence D. Merkle, and Gary B. Lamont
Air Force Institute of Technology
Wright-Patterson AFB, OH 45433
lmerkle, jolsan, dbrinkma, lamont@afit.af.mil

[*]Sponsored by the Air Force Office of Scientific Research and the Air Force Materiel Command's Wright Laboratories Materials Directorate

Comparison of Parallel Messy Genetic Algorithm Data Distribution Strategies

1 Introduction

The Schema Theorem provides a mechanism for theoretical analysis of the expected allocation of trials in a simple genetic algorithm (GA)[?]. It, along with the static building block hypothesis, gives rise to the identification of a class of problems called deceptive problems[?, ?, ?, ?]. Goldberg proposes the messy GA specifically to address the peculiarities associated with deceptive problems[?, ?, ?]. Validity of the SBBH, and therefore the significance of deception itself is the subject of some debate[?, ?, ?], which this paper does not attempt to address. However, the messy GA has performed well enough in comparison with the simple GA on non-deceptive problems to merit additional investigation[?]. Research in parallel genetic algorithms has focused primarily on identifying efficient methods for selection and communication strategies[?, ?, ?, ?]. Because the simple GA typically uses random initialization, data distribution has not been an issue in parallel GA research. In contrast, due to the use of the partially enumerative initialization scheme, parallelization of the messy GA demands the consideration of data distribution.

Analysis, which is confirmed by experiment[?], shows that the fraction of execution time required by the primordial phase increases significantly for larger problem sizes, and dominates the overall execution time. In turn, the vast majority of the execution time in the primordial phase is consumed by the tournament selection algorithm. The implication is that in order to obtain reasonable speedup from parallelization, initial effort must focus on efficient parallelization of tournament selection in the primordial phase. This paper describes four parallel implementations of the messy genetic algorithm, each of which uses a different strategy to exploit the data parallelism present in the primordial phase in order to obtain speedup in the tournament selection algorithm (Section 2). It also describes experiments comparing the execution time and solution quality of the four implementations and the results of those experiments (Section 3). Finally, it presents conclusions (Section 4) and recommendations (Section 5).

2 Primordial Phase Data Distribution Strategies.

In order to implement a parallel tournament selection algorithm, it is convenient to first describe the algorithm in an architecturally independent manner. Chandy and Misra propose UNITY (Unbounded Nondeterministic Iterative Transformations) as a method for obtaining such a description[?]. A UNITY program describes the requirements for a process without specifying the order of operations or the mapping of operations to processors. Thus, a UNITY program may be mapped to any architecture, whether it be sequential, asynchronous shared-memory, or distributed memory. The description of a mapping describes how the UNITY program is executed on the target architecture. Mappings for particular classes of architectures exhibit common characteristics. The target architecture in this study is a distributed memory (DM) system, which is described formally as consisting of a fixed set of processors, a fixed set of communication channels, and a memory for each processor[?, 83]. Using the formal requirement for mappings to DM architectures, and a UNITY representation of the

Nodes	0	1	2	3	4	5	6	7	8+
1	32480	-	-	-	-	-	-	-	0
2	19584	12896	-	-	-	-	-	-	0
4	10752	8832	7168	5728	-	-	-	-	0
8	5632	5120	4640	4192	3776	3392	3032	2696	0
16	3872	3552	3248	2960	2688	2432	2192	1968	9568
32	3248	3024	2808	2600	2400	2208	2024	1848	12320
64	3248	3024	2808	2600	2400	2208	2024	1848	12320

Table 1: Indexed Distribution Strategy Allocation

tournament selection algorithm, it has been proven that standard tournament selection is inherently sequential [**?**, pp. 78-79]. However, many approximations exist which are parallelizable. A modified algorithm may be obtained by first arbitrarily allocating the variables of the original algorithm to processors and then modifying the statements in many way such that each statement references only variables which are allocated to a single processor. The UNITY description for one such algorithm is shown at Figure 1. This algorithm may be directly mapped to a distributed memory architecture by allocating to the same processor those statements which share the same value of j. The number of solutions allocated to each processor by this algorithm is determined by the *SUB_POP_SIZE* data structure. The actual solutions which are assigned to the same subpopulation are determined by the mapping from the *distribution* data structure to the *pop* data structure. The remainder of this section presents four versions of the *Initialize_Population* process, each of which describes a different partitioning and mapping of solutions to the processors.

The first strategy, called the "indexed" strategy assigns building blocks to processors using an interleaving scheme, using the block's first defined locus as the interleaving key. Thus, a solution which is defined over loci $n_1, n_2, ...n_k$, so that the first defined locus is $n_{min} = min(n_1, n_2, ...n_k)$, will be allocated to processor $j = n_{min} \bmod m$, where m is the number of processors. Using this strategy each processor j, $0 \leq j < m$, is allocated

$$N_j = \sum_{i=0}^{I} \binom{l - j - im - 1}{k - 1} \tag{1}$$

solutions, where l is the string length, k is the block size, and

$$I = \lceil (l - j)/m \rceil - 1. \tag{2}$$

Thus, use of this strategy for a 30-bit problem with a block size of 3, allocates solutions to processors as shown in Table 1. This strategy allocates significantly more solutions to some processors than to others. In cases where the number of processors is greater than the string length, some processors are not allocated any solutions. In particular, for the above example, the strategy allocates no solutions to processors greater than 29. The second strategy, "modified indexed" distribution, is a modification of the first strategy. Solutions for which the first defined locus is greater than the number of processors are assigned to processors in reverse order. Thus, a solution which is defined on loci $n_1, n_2, ...n_k$ will be allocated to processor $j = n_{min}$ if $n_{min} < m$ or $j = m - 1 - n_{min} \bmod m$ if $n_{min} \geq m$. Using this strategy each processor j, $0 \leq j < m$, is allocated

$$N_j = \binom{l - j - 1}{k - 1} + \sum_{i=1}^{I} \binom{l - (i+1)m + j - 1}{k - 1} \tag{3}$$

```
Function $Conduct\_Parallel\_Tournament\_Selection$
declare
    $dist$ : array $[1..SUB\_POP\_SIZE[j]]$ of integer
    $perm$ : array $[1..SUB\_POP\_SIZE[j]]$ of integer
    $n_{sh}$ : integer
    $\theta$ : integer
    $popindex$ : integer
    $found$ : array $[1..SUB\_POP\_SIZE[j]]$ of boolean
    $tourn\_seq$ : integer
always
    $cand1 = dist[SUB\_POP\_SIZE[j] + popindex]$
    $cand2 = dist[SUB\_POP\_SIZE[j] +$
                $((popindex + i \bmod SUB\_POP\_SIZE[j]) + 1)]$
    $\theta = \left\lceil \frac{\lambda_1 \lambda_2}{l} \right\rceil$
    $compatible = \theta \leq |\ cand1.loci \cap cand2.loci\ |$
    $SUB\_POP\_START[j] = \sum_{i=0}^{j-1} SUB\_POP\_SIZE[j]$
initially
    $tourn\_seq = 0$
assign

      $\langle || \ \forall\ j : 1 \leq j \leq P$ ::
      {Initialization}
          $\langle || \ \forall\ i : 1 \leq i \leq SUB\_POP\_SIZE[j]$ ::
              $found[SUB\_POP\_START[j] + i]$ := FALSE
              $||\ perm[SUB\_POP\_START[j] + i]$
                  := $Random(POP\_SIZE)$
              $||\ n_{sh} := l$
              $||\ tourn\_seq := 1$
          $\rangle$ if $tourn\_seq = 0 \wedge conduct\_tournament$
          ||
      {Shuffle the population}
          $\langle\ []\ \forall\ i : 1 \leq i \leq SUB\_POP\_SIZE[j]$ ::
              $dist[SUB\_POP\_START[j] + i]$ :=
                  $dist[perm[SUB\_POP\_START[j] + i]]$
              $||\ dist\ [perm\ [SUB\_POP\_START[j] + i]]$ :=
                  $dist[SUB\_POP\_START[j] + i]$
              $||\ tourn\_seq := 2$
          $\rangle$ if $tourn\_seq = 1$
          ||
      {Conduct tournaments (nested loop)}
          $\langle ||\forall\ popindex : 1 \leq popindex \leq SUB\_POP\_SIZE[j]$ ::
              $\langle []\forall\ i : 1 \leq i \leq n_{sh} \wedge \neg found[popindex]$ ::
                  $found[popindex]$,
                  $newdist[popindex]$,
                  $tourn\_seq$ :=
                  TRUE, $cand1$, 3
                      if {First mate is more fit}
                      $(fitness(cand1) > fitness(cand2) \vee$
                      $(fitness(cand1) = fitness(cand2)) \wedge \lambda_1 < \lambda_2)$
                      $\wedge\ compatible)$
                  $\sim$
                  TRUE, $cand2$, 3
                      if {Second mate is more fit}
                      $(fitness(cand2) > fitness(cand1) \vee$
                      $(fitness(cand1) = fitness(cand2)) \wedge \lambda_1 > \lambda_2)$
                      $\wedge\ compatible)$
                  $\sim$
                  FALSE, $cand1$, 3
                      if$\neg\ compatible$
              $\rangle$
          $\rangle$ if $tourn\_seq = 2$
          ||
      {Copy new population}
          $\langle || \ \forall\ i : 1 \leq i \leq SUB\_POP\_SIZE[j]$ ::
              $dist[SUB\_POP\_START[j] + i]$ :=
                  $newdist[SUB\_POP\_START[j] + i]$
              $||\ conduct\_tournament$ := FALSE
              $||\ tourn\_seq := 0$
                      if $found[SUB\_POP\_START[j] + i]$
```

Nodes	0	1	2	3	4	5	6	7	Variation
1	32480	-	-	-	-	-	-	-	0
2	15624	16856	-	-	-	-	-	-	0
4	7536	7888	8296	8760	-	-	-	-	0
8	4096	4032	3992	3976	3992	4040	4120	4232	0
32	3248	3024	2808	2600	2408	2232	2072	1928	12160
32	3248	3024	2808	2600	2400	2208	2024	1848	12320
64	3248	3024	2808	2600	2400	2208	2024	1848	12320

Table 2: Modified Indexed Distribution Strategy Allocation

solutions. Thus, use of this strategy for the same problem allocates solutions to processors as shown in Table 2. This strategy allocates solutions more evenly than the indexed strategy, although there is still some imbalance. In cases where the number of processors is greater than half of the string length, this strategy approaches the indexed strategy.

The third strategy implicitly orders the building blocks, then interleaves the building blocks across the processors based upon their position in the ordering. This is the strategy used in Dymek's parallel implementation of the messy GA[**?**]. It is called the "interleaved" distribution strategy. The implicit ordering is such that if two solutions x and Y are defined on loci $x_1, x_2, ...x_k$ and $y_1, y_2, ...y_k$, where $x_1 < x_2 < ... < x_k$ and $y_1 < y_2 < ... < y_k$, solution x occurs first in the ordering if and only if there exist an x_i and y_i such that $x_i < y_i$ and for all $0 < j < i$, $x_i = y_i$. Each solution then has a unique index x such that it occurs after exactly x solutions in the ordering. Each solution is allocated to processor $j = x \bmod m$. Using this strategy each processor j, $0 \leq j < m$, is allocated

$$N_j = \left\lceil \frac{\begin{pmatrix} l \\ k \end{pmatrix} - j}{m} \right| \tag{4}$$

solutions. Thus, this strategy allocates solutions to processors for the example problem as shown in Table 3. This strategy allocates solutions as evenly as possible.

The last strategy uses the same implicit ordering, and assigns the building blocks to the processors using a block distribution strategy. Each solution is allocated to processor $N_j = \left\lfloor \frac{XN}{m} \right\rfloor$ where $N = \sum_j N_j$, is the total population size. This strategy also allocates the solutions as evenly as possible.

In order to anticipate the effect of distribution strategy on primordial phase execution time, some informal analysis of the behavior of the tournament selection algorithm is in order. The algorithm, as presented previously in Figure 1, contains a nested loop. For each member of the current subpopulation, the inner loop randomly selects solutions as potential mates. It continues selecting until either a solution is found which is "compatible" with the first mate or the shuffle size $n_s h$ has been exceeded[**?**]. Two solutions x and y are compatible if they are defined over loci $\{x_1, x_2, ...x_k\}$ and $\{y_1, y_2, ...y_k\}$, the intersection of which contains at least some threshold θ number of elements. The shuffle size is an input parameter specified at run time. Thus, the execution time of the primordial phase on a given processor is a function of

Nodes	0	1	2	3	4	5	6	7	Variation
1	32480	-	-	-	-	-	-	-	0
2	16240	16240	-	-	-	-	-	-	0
4	8120	8120	8120	8120	-	-	-	-	0
8	4060	4060	4060	4060	4060	4060	4060	4060	0
16	2030	2030	2030	2030	2030	2030	2030	2030	16240
32	1015	1015	1015	1015	1015	1015	1015	1015	24360
64	508	508	508	508	508	508	508	508	28416

Table 3: Interleaved/Block Distribution Strategy Allocations

- the number of solutions allocated to the processor's subpopulation,
- the probability with which two solutions randomly selected from a particular subpopulation are compatible, and
- the shuffle size.

The data distribution strategy affects both the number of solutions allocated to each subpopulation and the probability of compatibility. As discussed previously, the block and interleaved strategies allocate solutions to processors uniformly, while the indexed and modified indexed strategies do not. On this basis, the two former strategies should require less execution time that the latter two, especially in cases for which the number of processors is the same magnitude or larger than the string length. However, under either of the two indexed strategies, solutions are assigned to the same processor if their initial defining loci are the same. Assuming that the block size, k, and the string length l, are such that $k^2 \leq l$, any two such solutions are compatible. Thus, the probability of the compatibility on an indexed strategy is relatively high. Likewise, because of the implicit ordering used in the interleaved and blocked strategies, solutions which occur close to each other in the ordering are likely to be compatible. This implies that the probability of compatibility in the block strategy is relatively high, while that for the interleaved strategy is close to that of the sequential algorithm. On the basis of compatibility, the interleaved strategy should require greater execution time than the other three. With any of the four distribution strategies, the population following the primordial phase is likely to be very similar to that obtained in a sequential implementation, as long as the average non-zero subpopulation size is large compared with the shuffle size.

3 Results.

In order to determine the effects of each of the data distribution strategies on solution quality and execution time, a series of experiments are performed. Versions of the parallel messy genetic algorithm[**?**] which use modified data distribution strategies are implemented on a 64-node iPSC/i860 in C under the UNIX System V/386 Release 3.2 operating system[†]. In each case, the reduced subpopulations are recombined prior to the juxtapositional phase.

[†]Courtesy of the Intel Supercomputer Training Center, Beaverton, Oregon.

Subproblem	Loci		
1	1	6	11
2	2	7	12
3	3	8	13
4	4	9	14
5	5	10	15
6	16	21	26
7	17	22	27
8	18	23	28
9	19	24	29
10	20	25	30

Table 4: Order 3 Deceptive Function Subproblems

For each strategy, the problem solved is the order-3 fully deceptive binary functional optimization problem addressed by Dymek[**?**], which is a slight modification of the problem described by Goldberg[**?**]. The problem consists of ten 3-bit subproblems, each of which is order-3 fully deceptive. The total fitness of a solution to the full problem is the sum of the fitnesses of the solutions to the subproblems. The encoding scheme for the function is based on a string of thirty genes and a binary genic alphabet, as defined by Goldberg[**?**]. The bits corresponding to a particular subproblem are separated within the string, as shown in Table 4. Separation of the genes increases the defining length of important building blocks, thus making the deceptive problem GA-hard.

The 64-node iPSC/i860 allocates processors in sets of 1, 2, 4, 8, 16, 32, or 64. Each MGA implementation is executed 10 times for each of the seven possible hypercube dimensions, using 10 randomly generated seeds. The same ten seeds are used for all strategies and all hypercube dimensions. The number of experiments is selected to be large enough to give a reasonable chance of obtaining results which are statistically significant at the 1% level. A total of four reductions are performed, with a reduction interval of 2. The shuffle number is 30, the cut probability is 0.0166667, and the splice probability is 1.0. A total of 19 primordial and juxtapositional generations are performed, and the overflow factor is 1.6.

The average solution quality obtained using each of the four distribution strategies for each hypercube dimension is shown at Table 5. The data are compared statistically using the Kruskal-Wallis H Test, which does not require that the data obey a normal distribution [**?**, p. 544]. Tests at the 5% level of significance for each hypercube dimension indicate that choice of distribution strategy does not have a statistically significant effect on solution quality for the fully deceptive binary function except that for 32 or more processors the interleaved strategy does not perform as well as the other three strategies.

Likewise, the average execution time for each of the four distribution strategies for each hypercube dimension is shown at Table 6. The data are again compared statistically using the Kruskal-Wallis H Test. Tests at the 0.5% level of significance for each hypercube dimension indicate that choice of distribution strategy has a statistically significant effect on execution time. The exception is that the execution times for the indexed strategy on 32 or more processors and the modified indexed strategy on 16 or more processors are not significantly different. Average speedups are shown for each implementation for the primordial phase

	Distribution Strategy			
Processors	Index	ModIndex	Block	Interleave
1	297.8	297.8	297.2	297.2
2	298.2	297.0	298.0	298.0
4	297.6	298.0	298.2	298.2
8	296.6	296.8	297.8	297.8
16	296.4	298.4	299.0	298.0
32	299.2	299.2	298.8	296.8 (*)
64	299.2	299.2	298.4	296.0 (*)

Table 5: Solution Quality
(*) Indicates *presence* of statistically significant difference

	Distribution Strategy			
Processors	Index	ModIndex	Block	Interleave
1	2.016e1	2.003e1	1.951e1	1.952e1
2	8.891e0	8.733e0	8.228e0	9.791e0
4	4.080e0	3.649e0	3.706e0	5.130e0
8	1.854e0	1.395e0	1.540e0	2.570e0
16	9.637e-1	5.907e-1 (*)	7.287e-1	1.229e0
32	5.911e-1 (*)	5.906e-1 (*)	3.496e-1	6.186e-1
64	5.910e-1 (*)	5.912e-1 (*)	2.086e-1	3.379e-1

Table 6: Execution Time
(*) Indicates *absence* of statistically significant difference

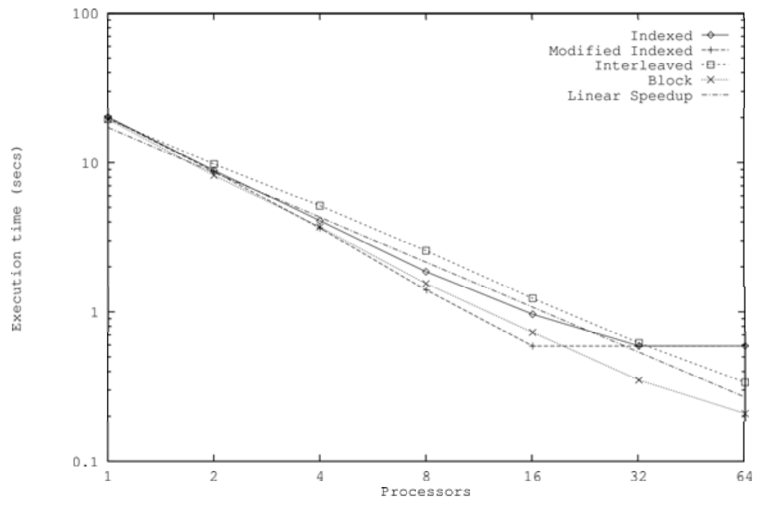

Figure 2: Primordial Phase Speedup

alone (Figure 2) and the overall execution (Figure 3). The absence of speedup in overall execution time for greater than 16 processors indicates that additional speedup must be obtained from parallelization of the juxtapositional phase. For 8 or fewer processors, the modified indexed strategy yields the best speedup of the primordial phase, while the block strategy results in the next best speedup. As expected, the modified indexed strategy exhibits significantly worse speedup for implementations for which the number of processors exceeds one half of the string length. The block strategy obtains the best primordial phase speedup in such implementations.

The indexed, modified indexed, and block distribution strategies all result in “super-linear speedup” of the primordial phase. The modified indexed and block distribution strategies also result in “super-linear speedup” of the initialization, primordial phase, and conversion operations together. The presence of “super-linear speedup” is misleading in that the parallel algorithm is not completely functionally equivalent to the sequential algorithm. The modifications introduced in the parallelization account for part of the speedup. Application of the same modifications to the sequential algorithm should result in reduced execution time.

4 Conclusions.

The execution time of the MGA is dominated by the primordial phase, indicating that significant speedup requires parallelization of the tournament selection algorithm. The algorithm is inherently sequential due to the property that any tournament may involve any member of the population. Parallel algorithms exist which approximate the behavior of the sequential tournament selection algorithm. The initial population may be distributed in a number of ways, including the “indexed,” “modified indexed,” “interleaved,” and “block” distribution strategies. The indexed and modified indexed strategies allocate solutions to processors based upon the solution’s first defined locus, resulting in uneven distribution. The interleaved and block distribution strategies result in even distributions.

Primordial phase execution time is a function of the number of solutions in the subpop-

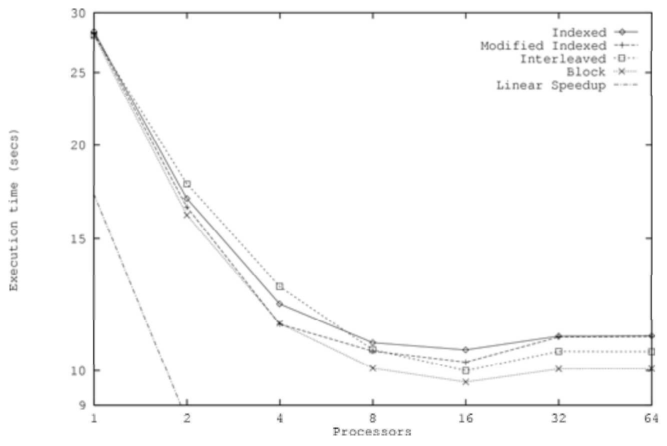

Figure 3: Overall Speedup

ulation, the probability that two randomly selected solutions are compatible, and the shuffle size. The distribution strategy affects the number of solutions in the subpopulation and the probability that two solutions are compatible. Experiments comparing the solution quality and execution time of the four distribution strategies when applied to the fully deceptive binary function show that in most cases the distribution strategy does not have a significant effect on solution quality.

In contrast, results indicate that the distribution strategy does have a significant effect on primordial phase execution time. The number of solutions allocated to each processor is one factor influencing execution time, but not the only one. The degree of compatibility among solutions assigned to the processor also contributes significantly to execution time in the primordial phase. Where the degree of compatibility is high, as with the indexed, modified indexed, and block strategies, the second mate for each tournament is found relatively quickly. Conversely, where the degree of compatibility is low, as with the interleaved strategy, additional time is required to find a compatible mate. The overall effect is that the interleaved strategy results in significantly higher execution time in the primordial phase than the other strategies.

5 Recommendations.

These results are explained by an informal theoretical analysis of the behavior of the parallel tournament selection algorithm. A more complete and rigorous theoretical treatment is needed to generalize the results to other problems and architectures. Also, additional experimentation is needed. Experiments involving larger instances of similar problems and larger architectures are needed to test the scalability of the results. Experiments involving other types of problems, such as permutation problems and difficult but non-deceptive function optimization problems, are needed to ensure that the results are not peculiar to the pedagogical function used in this study. Of critical importance is the identification of new distribution strategies which scale to arbitrarily large architectures while still respecting compatibility requirements.

References

[1] Allen, Arnold O. *Probability, Statistics, and Queueing Theory: With Computer Science Applications*. Computer Science and Scientific Computing, San Diego, California: Academic Press, Inc., 1990.

[2] Bethke, Albert D. *Genetic Algorithms as Function Optimizers*. PhD dissertation, The University of Michigan, Ann Arbor MI, 1980.

[3] Chandy, K. Mani and Jayadev Misra. *Parallel Program Design: A Foundation*. Reading MA: Addison-Wesley Publishing Company, 1989.

[4] Cohoon, J. P. and others. "A multi-population genetic algorithm for solving the k-partition problem on hypercubes." *Proceedings of the Fourth International Conference on Genetic Algorithms*. 244–248. San Mateo CA: Morgan Kaufmann Publishers, Inc., 1991.

[5] Dymek, Capt Andrew. *An Examination of Hypercube Implementations of Genetic Algorithms*. MS thesis, AFIT/GCE/ENG/92-M, Air Force Institute of Technology School of Engineering, Wright-Patterson AFB OH, March 1992 (AD-A248092).

[6] Forrest, Stephanie and Melanie Mitchell. "Relative Building-Block Fitness and the Building-Block Hypothesis." *Foundations of Genetic Algorithms 2 (FOGA-2)*, edited by D. Whitley. San Mateo, California: Morgan Kaufmann, 1992.

[7] Goldberg, David E. "Simple Genetic Algorithms and the Minimal Deceptive Problem." *Genetic Algorithms and Simulated Annealing* edited by Lawrence Davis, London: Pitman, 1987.

[8] Goldberg, David E. "Genetic Algorithms and Walsh Polynomials: Part I, A Gentle Introduction," *Complex Systems*, *3*:129–152 (1989).

[9] Goldberg, David E. "Genetic Algorithms and Walsh Polynomials: Part II, Deception and its Analysis," *Complex Systems*, *3*:153–171 (1989).

[10] Goldberg, David E. and others. "Messy Genetic Algorithms: Motivation, Analysis, and First Results," *Complex Systems*, *3*:493–530 (1989).

[11] Goldberg, David E. and others. "Messy Genetic Algorithms Revisited," *Complex Systems*, *4*:415–444 (1990).

[12] Goldberg, David E. and others. "Don't Worry, Be Messy." *Proceedings of the Fourth International Conference on Genetic Algorithms*. 24–30. San Mateo CA: Morgan Kaufmann Publishers, Inc., 1991.

[13] Grefenstette, John J. "Deception Considered Harmful." *Foundations of Genetic Algorithms 2 (FOGA-2)*, edited by D. Whitley. San Mateo, California: Morgan Kaufmann, 1992.

[14] Holland, John H. *Adaptation in Natural and Artificial Systems* (First mit press Edition). Cambridge, MA: MIT Press, 1992.

[15] Merkle, Laurence D. *Generalization and Parallelization of Messy Genetic Algorithms and Communication in Parallel Genetic Algorithms.* MS thesis, Air Force Institute of Technology, WPAFB OH 45433, December 1992.

[16] Pettey, Chrisila B. and Michael R. Leuze. "A theoretical investigation of a parallel genetic algorithm." *Proceedings of the Third International Conference on Genetic Algorithms.* 398–405. San Mateo CA: Morgan Kaufmann Publishers, Inc., 1989.

[17] Spiessens, Piet and Bernard Manderick. "A Massively Parallel Genetic Algorithm: Implementation and First Analysis." *Proceedings of the Fourth International Conference on Genetic Algorithms.* 279–285. San Mateo CA: Morgan Kaufmann Publishers, Inc., 1991.

[18] Tanese, Reiko. "Parallel genetic algorithms for a hypercube." *Genetic Algorithms and Their Applications: Proceedings of the Second International Conference on Genetic Algorithms.* 170–176. Hillsdale NJ: Lawrence Erlbaum Associates, Inc., 1987.