Empirical Investigations on Parallelized Linkage Identification

Masaharu Munetomo[1], Naoya Murao[2], and Kiyoshi Akama[1]

[1] Information Initiative Center
Hokkaido University, Sapporo 060-0811, Japan
[2] Graduate School of Engineering,
Hokkaido University, Sapporo 060-0811, Japan

Abstract. To solve GA-difficult problems in which we cannot ensure tight linkage in their encoding, advanced methods such as linkage identification techniques and estimation of distribution algorithms work effectively although they need some additional computational cost. The computation time can be reduced by employing parallel computers and several approaches have been proposed for their parallelized algorithms. This paper presents empirical results on parallelization of the linkage identification compared to that of an estimation of distribution algorithm.

1 Introduction

Among a series of advanced methods in genetic and evolutionary computations, linkage identification/learning techniques and estimation of distribution algorithms (EDAs) are considered two major approaches to realize effective mixing of building blocks even when tight linkage is not ensured. In linkage identification techniques such as the linkage identification by nonlinearity check (LINC)[3, 2, 4], a linkage group – a set of loci tightly linked to form a building block (BB) – can be detected by checking nonlinearity with perturbations on a pair of loci. Effective mixing is realized by applying crossovers based on the identified linkage groups. In order to obtain accurate solutions with less computational time, several approaches have been proposed for their parallelization. In parallel linkage identification originally proposed by Munetomo et. al.[5], calculation of linkage identification for each pair of loci is highly independent each other and therefore, it is easy to parallelize. In numerical simulations comparing the number of fitness evaluations, the parallelized linkage identification achieves quasi-linear speedup that exceeds conventional parallel GAs such as island models. They also show that overall performance of the parallel linkage identification exceeds conventional methods not only when we cannot ensure tight linkage, but also when fitness contribution of BBs is largely different each other.

On the other hand, estimation of distribution algorithms (EDAs) such as the Bayesian optimization algorithm (BOA)[6, 7] estimate probability distribution of strings after applying selections to generate proper candidates of BBs in the next generation. The model building process in BOA by constructing Bayesian

© Springer-Verlag Berlin Heidelberg 2004

networks needs extensive computational cost, which should be parallelized. In the distributed BOA (DBOA) proposed in Ocenasek's doctoral thesis[1], each processor generates its subpopulation independently and model-building process is also parallelized by assigning nodes that represent probabilistic variables of a Bayesian network to processors.

In this paper, we perform empirical investigations on parallel linkage identification in comparison to the parallelized BOA. This is because these two approaches can solve difficult problems where we cannot ensure tight linkage in advance, whereas, other conventional PGAs such as with master-slave and island models cannot solve loosely-linked difficult problems even when we employ a number of parallel processors. (In the original paper of parallel linkage identification[5], results of numerical simulations on tightly encoded easier problems are presented in comparison to the conventional PGAs.)

The experiments are performed on an actual parallel architecture not by numerical simulations, because comparisons based only on the number of fitness evaluations do not reflect actual performance, especially for the Bayesian optimization algorithm whose majority of overheads lie in its model-building process which does not need explicit fitness evaluations.

2 Parallel Linkage Identification

Linkage identification by nonlinearity check (LINC) [3, 2, 4] is a first step toward identifying linkage groups – sets of loci tightly linked – only by bit-wise perturbations. In the LINC, linkage groups are identified by applying perturbations for pairs of loci to detect nonlinear interactions in fitness changes. In order-k delinearble functions – the sum of nonlinear sub-functions whose maximum order is k, nonlinear interactions exist only inside the sub-functions. When we assume that the objective function of the problem is a strict or loose order-k delinearble function, we can identify bit-positions for each sub-function by detecting nonlinearity by the perturbations. The nonlinearity detection must be performed on all the strings in a properly-sized population because even in a GA-difficult nonlinear sub-function, some linear attractors might be existent.

Therefore, the LINC calculates the following values for each pair of loci (i, j) and for all the strings s in a randomly initialized population that has $O(c2^k)$ strings where $c = \log(1-r)$ is a constant determined by the allowable probability r of detection failure, and k is the maximum order of linkage groups:

$$\begin{aligned} \Delta f_i(s) &= \ f(..\bar{s_i}.....) - f(..s_i.....) \\ \Delta f_j(s) &= \ f(.....\bar{s_j}..) - f(.....s_j..) \\ \Delta f_{ij}(s) &= f(..\bar{s_i}.\bar{s_j}..) - f(..s_i.s_j...), \end{aligned} \quad (1)$$

where $f(s)$ is a fitness functions of a string s and $\bar{s_i} = 1 - s_i$ is a bitwise perturbation ($0 \to 1$ or $1 \to 0$) at the i-th locus of s.

If $\Delta f_{ij}(s) \neq \Delta f_i(s) + \Delta f_j(s)$, then s_i and s_j are considered to belong to a same linkage group – we add i to the linkage group of locus j, and add j to the linkage group of i. Otherwise, s_i and s_j may not be a member of a linkage group, or they are linked but linearity exists in the current context. The LINC checks the nonlinearity condition for all the string s in a population. When the nonlinearity condition is satisfied for at least one string s, the pair (i, j) should be considered tightly linked. The LINC is performed according to the following algorithm[3]:

1. Randomly initialize a population of $O(c2^k)$ binary strings
2. For each string in the population, performs the following
3. For each pair of loci (i, j), performs the following
4. Calculate $\Delta f_{ij}(s)$, $\Delta f_i(s)$, and $\Delta f_j(s)$ by bit-wise perturbations
5. If $\Delta f_{ij}(s) \neq \Delta f_i(s) + \Delta f_j(s)$, locus i is included in the linkage group of j and locus j is included in the linkage group of i

The underlying assumption of the LINC is that fitness functions can be represented as the sum of nonlinear sub-functions whose optimal sub-solutions are building blocks. Strict nonlinearity condition cannot be applicable to real-world problems, therefore, we have introduced an allowable error ε for the condition and modified the above nonlinearity as follows: if $|\Delta f_{ij}(s) - (\Delta f_i(s) + \Delta f_j(s))| > \varepsilon$ then loci i and j is considered tightly linked.

The computational cost of the LINC is $O(nl^2)$ where n is the population size of $O(c2^k)$ (k : maximum order of building blocks) and l is the string length. The nonlinearity check for each pair of loci can easily be parallelized because there is no interdependency among their calculations. Employing P processors, we can reduce its computation time to $O(nl^2/P)$.

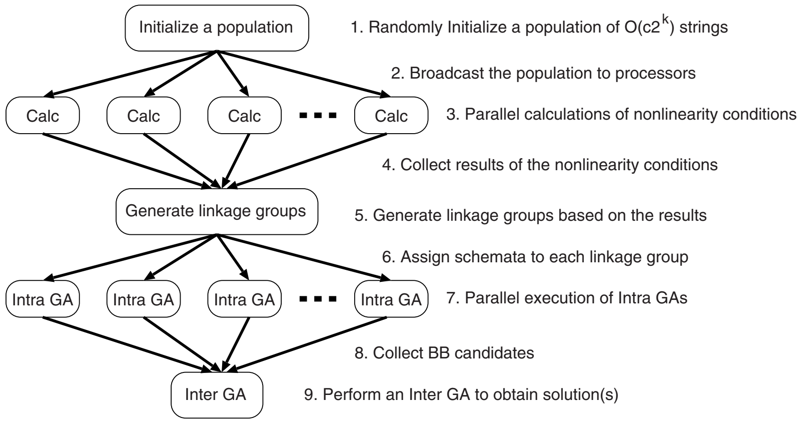
Initialize a population
1. Randomly Initialize a population of O(c2^k) strings
2. Broadcast the population to processors
Calc
Calc
Calc
Calc
3. Parallel calculations of nonlinearity conditions
4. Collect results of the nonlinearity conditions
Generate linkage groups
5. Generate linkage groups based on the results
6. Assign schemata to each linkage group
Intra GA
Intra GA
Intra GA
Intra GA
7. Parallel execution of Intra GAs
8. Collect BB candidates
Inter GA
9. Perform an Inter GA to obtain solution(s)

Fig. 1. PGA based on linkage identification[5].

The parallelized LINC (pLINC) is performed on a master-slave parallel computer consisting of a master processor and slave processors as follows:

1. A master processor randomly initializes a population of $n = O(c2^k)$ strings and broadcasts them to all the slave processors. (Therefore, all the processors have the same initial populations.)
2. Each slave processor calculates its assigned nonlinearity conditions for each pair of loci.
3. Results of the condition are collected from slave processors to the master processor.
4. The master generates linkage groups based on the collected results.
5. The master processor divides strings into schemata according to the obtained linkage groups and assigns them to the slave processors.
6. Each slave processor performs Intra GA that is performed inside a linkage group to search building block candidates concerning the assigned linkage group.
7. The master collects building block candidates obtained in slave processors.
8. The master performs Inter GA for the candidates to obtain solutions.

The Intra GAs and the Inter GA were originally introduced in a GA based on linkage identification[2]. For Intra GA, initial strings are divided into schemata based on the obtained linkage groups. Figure 2 (left) illustrates this decomposition process.

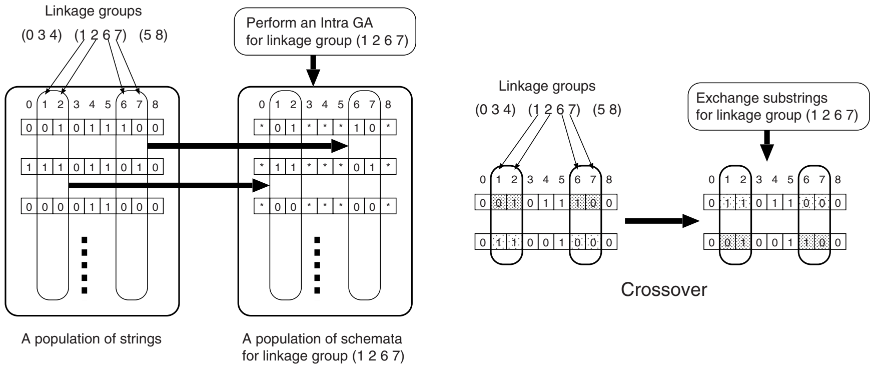

Fig. 2. Division of strings into schemata based on linkage groups (left) and Crossover operator of the Inter GA (right) [2].

The Intra GA performs a SGA with ranking selections, uniform crossovers and simple mutations inside a linkage group. To evaluate schemata, a template based on the current best string is employed, which is similar to competitive templates originally introduced in messy GA[8]. After the Intra GAs, we select a limited number of well-performed schemata as BB candidates in each linkage group.

The Inter GA applies ranking selections and crossovers based on the linkage groups that mix the BB candidates. Figure 2 (right) illustrates the crossover operator employed in the Inter GA.

3 Parallel Bayesian Optimization Algorithm

Estimation of distribution algorithms (EDAs) estimate probabilistic distributions of alleles for promising strings survived after applying selections, which is employed to make offsprings for the next generations. The early EDA models such as the PBIL (Population-Based Incremental Learning) estimates probability of 1's occurrence in each locus and generates offsprings based on the probabilities. A series of EDA methods have been proposed such as the UMDA (Univariate Marginal Distribution Algorithm), the BMDA (Bivariate Marginal Distribution Algorithm), and so on. The Bayesian optimization algorithm (BOA) proposed by Pelikan et. al[6, 7] is considered one of the most sophisticated algorithm among the EDA methods, which builds probabilistic models employing Bayesian networks. As for computational complexity of BOA, its network construction – searching an optimal network that minimizes the Bayesian-Dirichlet (BD) metric – tends to dominate its computational overhead, which should be parallelized. Computational complexity for searching network structure of BOA is $O(k2^k l^2 N + kl^3)$ where k is the maximum degree of nodes (maximum number of links connected to a node) and N is the number of instances.

Distributed BOA (DBOA)[1] parallelizes processes of generating strings and constructing Bayesian networks. Parallelization of the strings generation is simple: each node generates its assigned subpopulation of strings based on the conditional probabilities determined by Bayesian networks. In order to parallelize construction of the networks, each node of a network is assigned to each processor, which represents a random variable for a locus. Parallelized edge addition may cause inconsistency for the overall network – loops may be created because decision of adding edges is performed independently in each processor. In order to avoid loops, DBOA introduces a permutation of nodes: an edge from a source node can only be connected to one which is after the source in the permutation. For example, when we specify a permutation of nodes such as (2 3 6 1 5 4), an edge from node 6 can only be connected to nodes 1, 5, and 4. This mechanism restricts search space of the network; therefore, the permutation is randomly regenerated in each stage of the network construction to have an optimal network. The time complexity of the parallelized version of the network construction is approximated by $O((k2^k l^2 N + kl^3)/P)$ where P is the number of processors.

4 Empirical Results

In order to compare the parallel LINC (pLINC) and the Distributed BOA (DBOA), numerical experiments comparing the number of fitness evaluations are not considered appropriate because they differ in their major source of computational overheads. The computational overheads of pLINC come from $O(l^2)$ fitness evaluations and those of DBOA come from $O(l^3)$ network construction that does not need fitness evaluations. To compare overall computational overheads, we need to observe those for fitness evaluations and for network construction in actual situations. Therefore, we perform experiments on an actual parallel architecture. In the following experiments, we employ a parallel computer SGI Onix 300 consisting of MIPS R14000/600MHz × 32 CPUs with 16GB shared memory connected via NUMAflex^{TM} high-speed network. For communications among processors, Message Passing Interface (MPI) is employed.

We employ an one-max function and the sum of k-bit trap functions defined as follows as our test functions.

$$f(s) = \sum_{i=1}^{L} f_i(u_i). \tag{2}$$

where f_i is a k-bit trap sub-function defined as follows:

$$f_i(u_i) = \begin{cases} k - u_i - 1 & \text{if } 0 \leq u_i \leq k-1 \\ k & \text{if } u_i = k \end{cases} \tag{3}$$

where u_i is the number of 1's occurrence in a k-bit substring s_i $(s = s_1 s_2 \cdots s_L)$.

We employ one-max, 3-bit trap, and 5-bit trap functions in the following experiments in order to control difficulty of the test problems. The string length of the problems is fixed to $l = 105$ ($L = 35$ for the 3-bit trap function and $L = 21$ for the 5-bit trap function). We observe time to obtain optimal solutions changing the number of processors $P = 1, 2, 4, 8, 16$. Population size and other parameters of the algorithms are optimized for each experiment in order to minimize the time to obtain optimal solutions, and we perform 20 experiments and plot their average in the following figures.

4.1 Comparison of Speedups

Figure 3 shows a comparison of speedups by pLINC and DBOA for the 5-bit trap test function. In the figure, the x-axis shows the number of processors employed and the y-axis is the speedup factor $S = T_s/T_p$ where T_s is execution time for a serial processor (when $P = 1$) and T_p is that for the parallel machine.

This illustrates pLINC achieves better results compared to DBOA in the speedup factors. This is because parallel calculation of nonlinearity condition for each pair of loci is highly independent and communication is necessary only when the master processor distributes an initial population and collects the results. On the other hand, parallel network construction in DBOA needs more frequent communications among processors during its search for an optimal network that minimizes the BD metric. This figure shows that pLINC is more preferable than DBOA in parallel computing environment for this problem. We have obtained similar results for one-max and 3-bit trap test functions. Since comparisons only by speedup factors are not enough to compare parallel algorithms – an inefficient algorithm by serial processing may achieve high speedup factors because its T_s is large, we perform comparisons for time to obtain optimal solutions in the following.

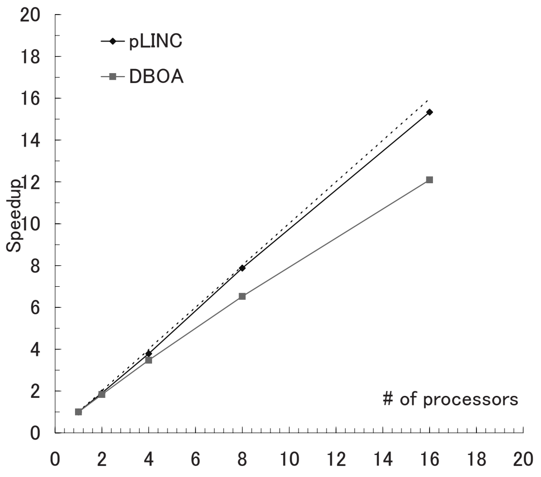

Fig. 3. Comparison of speedups between pLINC and DBOA.

4.2 Time for Linkage Identification vs. Model Building

Figure 4 shows comparisons of execution time for the one-max problem. In the figure, the x-axis is the number of processors employed and the y-axis shows execution time to obtain optimal solutions. For pLINC, time to evaluate fitness values dominates its computational cost. This is because it needs $O(l^2)$ fitness evaluations. On the other hand, for DBOA, model-building dominates its computational overheads. Comparing overall computation time, pLINC needs less than one-third of time for DBOA.

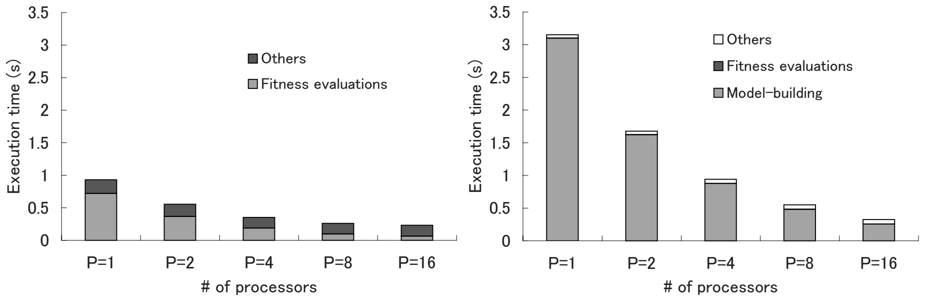

Fig. 4. Comparison of execution time to obtain optimal solutions between pLINC(left) and DBOA(right) for the one-max problem.

Figure 5 is the experimental results for the 3-bit trap function and figure 6 is for the 5-bit trap function. These results are similar to that for the one-max function. For more difficult 5-bit trap function, DBOA needs much more computational time than that of pLINC because computational overheads in

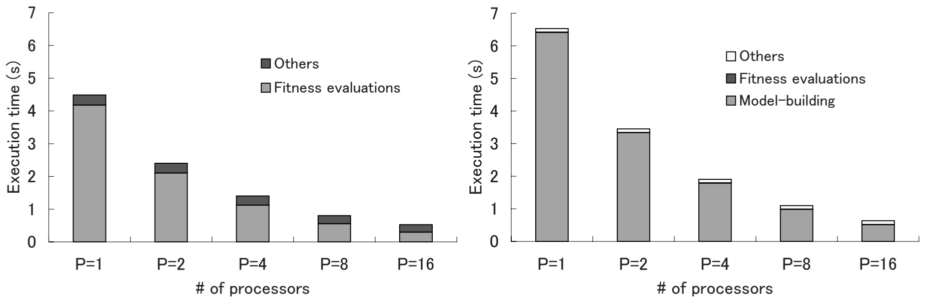

Fig. 5. Comparison of execution time to obtain optimal solutions between pLINC(left) and DBOA(right) for the 3-bit trap function.

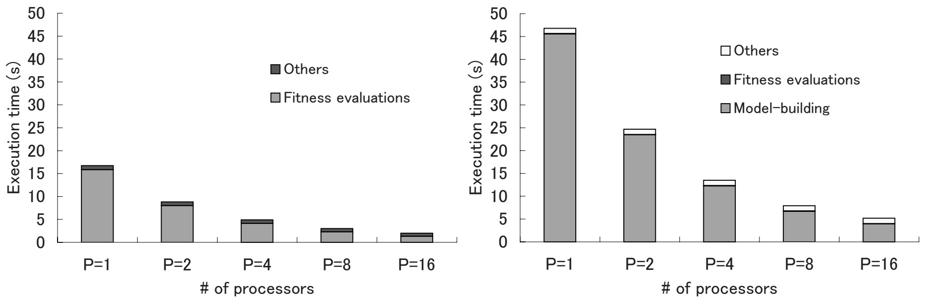

Fig. 6. Comparison of execution time to obtain optimal solutions between pLINC(left) and DBOA(right) for the 5-bit trap function.

its model-building process increase rapidly when problem difficulty increases concerning the order of BBs.

As expected from the previous results on speedups, fitness evaluations in pLINC and model-building in DBOA can be parallelized effectively – their computation time can greatly be reduced by employing parallel processors.

4.3 Effects of Fitness Evaluation Overheads

The major source of computational overheads for pLINC comes from its $O(l^2)$ fitness evaluations and that for DBOA comes from its network construction in model-building process. Therefore, it is expected that pLINC needs more computational time when time to evaluate a fitness value increases. In figure 7, we compare overall execution time employing 16 processors ($P = 16$) changing time to evaluate a fitness value by inserting wait in each evaluation. The 5-bit trap function is employed for this experiment.

From this result, computation time of the pLINC increases more rapidly compared with the DBOA along the wait increases. When wait equals to 1,000 micro-seconds = 1 milli-second, their computation time becomes about the same. This means we should employ pLINC when time to evaluate a fitness value is less

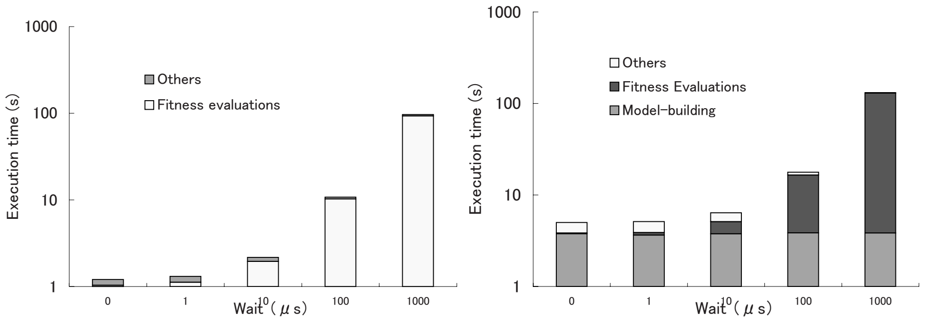

Fig. 7. Comparison of execution time varying waits for fitness evaluation time between LINC(left) and BOA(right) for the 5-bit trap function.

than 1 milli-second and otherwise we should employ DBOA in this experimental setting. A CPU of the parallel machine can perform around 500,000 floating-point calculations in 1 milli-second since a rough performance estimation of the R14000 is around 500MFLOPS (for a matrix calculation). This means that unless we need to perform exhaustive computations for a fitness evaluation, we should employ parallel linkage identifications.

5 Conclusions

Through empirical investigations performed on an actual parallel architecture, we show the effectiveness of the parallelized linkage identification compared with a parallel BOA. The result shows that pLINC achieves effective speedups compared to DBOA unless we need exhaustive computational overheads in evaluating a fitness value. In our experimental setting, the time border whether we should employ pLINC or DBOA is around 1 milli-second per evaluation, where we can perform about half million floating-point calculations for the processor we used.

We can consider the following source of GA-difficulties: (1) search space is extremely large such as in combinatorial optimization problems, (2) maximum order of building blocks is large – building block search becomes difficult and we need to have large initial populations, and (3) fitness evaluations are costly that require much computational overheads. The parallelized linkage identification is considered an effective approach that solves difficulties of the first and the second situations. For future works, we need to make theoretical models of relations between the source of problem difficulties and parallel algorithms to be employed.

References

1. Ocenasek Jiri. *Parallel Estimation of Distribution Algorithms*. PhD thesis, Brno University of Technology, 2002.
2. Masaharu Munetomo and David E. Goldberg. Designing a genetic algorithm using the linkage identification by nonlinearity check. Technical Report IlliGAL Report No.98014, University of Illinois at Urbana-Champaign, 1998.

3. Masaharu Munetomo and David E. Goldberg. Identifying linkage by nonlinearity check. Technical Report IlliGAL Report No.98012, University of Illinois at Urbana-Champaign, 1998.
4. Masaharu Munetomo and David E. Goldberg. Identifying linkage groups by nonlinearity/non-monotonicity detection. In *Proceedings of the 1999 Genetic and Evolutionary Computation Conference*, 1999.
5. Masaharu Munetomo, Naoya Murao, and Kiyoshi Akama. A parallel genetic algorithm based on linkage identification. In *Proceedings of the 2003 Genetic and Evolutionary Computation Conference, Part-1, LLNCS-2723*, pages 1222–1233, 2003.
6. M. Pelikan, D. E. Goldberg, and E. Cantú-Paz. BOA: The Bayesian optimization algorithm. IlliGAL Report No. 99003, Urbana, IL, 1999.
7. Martin Pelikan, David E. Goldberg, and Kumara Sastry. Bayesian optimization algorithm, decision graphs, and occam's razor. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, pages 519–526. Morgan Kaufmann, 2001.
8. David E. Goldberg, Kalyanmoy Deb, and Bradley Korb. Messy genetic algorithms revisited: Studies in mixed size and scale. *Complex Systems*, 4:415–444, 1990.