Solving the Error Correcting Code Problem with Parallel Hybrid Heuristics

Enrique Alba
Departamento de Lenguajes y Ciencias de la Computación
Universidad de Málaga
SPAIN
eat@lcc.uma.es

J. Francisco Chicano
Departamento de Lenguajes y Ciencias de la Computación
Universidad de Málaga
SPAIN
chicano@lcc.uma.es

ABSTRACT

Some telecommunication systems can not afford the cost of repeating a corrupted message. Instead, the message should be somewhat "corrected" by the receiver. In these cases an *error correcting code* is suitable. The problem of finding an error correcting code of n bits and M codewords that corrects a given maximum number of errors is NP-hard. For this reason the problem has been solved in the literature with heuristic techniques such as Simulated Annealing and Genetic Algorithms. In this paper we present a new local search algorithm for the problem: the Repulsion Algorithm. We further use a hybrid between Parallel Genetic Algorithm and this new algorithm to solve the problem, and we compare it against a pure Parallel Genetic Algorithm. The results show that an important improvement is achieved with the inclusion of the Repulsion Algorithm and the parallelism.

Categories and Subject Descriptors

I.2.8 [**Artificial Intelligence**]: Problem Solving, Control Methods, and Search—*Heuristic methods*; H.1.1 [**Models and Principles**]: Systems and Information Theory—*Information theory*

General Terms

Algorithms, Experimentation, Performance

Keywords

Information theory, Heuristics, Local search, Parallelism

1. INTRODUCTION

Some telecommunication systems can not afford the resubmission of a message that has been corrupted along the way. For these systems there are many kinds of codes that admit error correction in the receiver part; we focus here on binary linear block codes. Such codes are formed by a set of codewords, each of which is a binary string of the same length. The messages are composed of a sequence of such codewords. When a bit of a codeword has changed along the way, we say that there is an error in that codeword. If the Hamming distance between any pair of codewords is greater than or equal to $2e + 1$, then binary strings with up to e errors can be corrected without retransmission. Such correction is achieved by assigning to the incorrect binary string its nearest correct codeword.

Thus, a higher minimum Hamming distance between the codewords means a larger autocorrection capacity of the receiver. Given the length of the codewords and the number of words of a code the problem of finding an optimum code is NP-hard [5]. Exact techniques may be useful in small instances, but in realistic cases they are inefficient (do not scale). For this reason we must use heuristic techniques such as Simulated Annealing (SA) or Genetic Algorithm (GA) [5, 8].

The contribution of this paper is to present a new local search algorithm to solve the Error Correcting Code Problem (ECC). This algorithm has been called Repulsion Algorithm (RA). The algorithm is inspired in Physics and it is based on the movement of charged particles in the surface of a sphere. We solve the ECC with our local search algorithm and also with parallel and sequential GAs. Hybrid algorithms involving GA and RA are also discussed. In these hybrid algorithms the traditional mutation has been replaced by the RA. The results show that an important improvement in numerical efficiency can be achieved by using a new parallel GARA algorithm.

The paper is organized as follows. In Section 2 the class of ECC problems is presented. Then we make a brief introduction to parallel GAs in Section 3. The new RA is presented in Section 4. In Section 5 we describe the experiments performed and the results obtained for the ECC problem. Finally, in Section 6 we present our conclusions and point out future research issues.

2. THE ERROR CORRECTING CODE DESIGN PROBLEM (ECC)

As a basic step in building a communication system, the designer must solve the problem of finding a suitable code for transmitting messages through a noisy channel as reliably and quickly as possible. As it is often the case in optimization, we must fulfill two conflicting requirements to

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
SAC '04, March 14-17, 2004, Nicosia, Cyprus
Copyright 2004 ACM 1-58113-812-1/03/04 ...$5.00.

reach this objective. A first goal is to find minimum length codewords, so that we can ensure quick transmission of the messages encoded by using these codewords. On the other hand, the Hamming distance between codewords should be maximized, in order to guarantee a high level of error correction in the receiver part, what suggests including high redundance (larger codewords). A smaller length of the codewords means a smaller minimum Hamming distance between codewords. The underlying idea is the following: if all codewords are separated by d bits, then any modification of at most $\lfloor (d-1)/2 \rfloor$ bits in a valid codeword can be easily reverted by taking the most similar codeword.

A binary linear block code can be represented as a three-parameter vector (n, M, d), where n is the number of bits in each word in the code, M is the number of words in the code, and d is the minimum Hamming distance between any pair of words $C(i)$ and $C(j)$ $(i \neq j)$ in the code.

An optimal code is one that maximizes d, given n and M. In this paper we solve the problem for a 24-word code, with 12-bit length codewords ($M = 24, n = 12$) as in [5]. This instance of the problem has been largely addressed in the literature because of its difficulty. In [5], between 256 and 16384 processing elements of a SIMD machine are used to solve it. In [2], this instance is tackled with distributed GAs and sequential GAs, concluding on the superiority of the decentralized distributed approach. For this instance, the maximum Hamming distance (optimum) that can be reached is $d = 6$ [1].

3. PARALLEL GENETIC ALGORITHMS

Genetic Algorithms (GAs) are stochastic search methods that have been successfully applied in many search, optimization, and machine learning problems [4]. Unlike most other optimization techniques, GAs maintain a population of encoded tentative solutions that are competitively manipulated by applying some variation operators to find a global optimum.

A GA proceeds in an iterative manner by generating new populations of strings from the old ones. Every string is the encoded (binary, real, ...) version of a tentative solution. An evaluation function associates a fitness value to every string indicating its suitability to the problem. The canonical algorithm applies stochastic operators such as selection, crossover, and mutation on a random initial population, in order to compute a whole generation of new strings.

For non-trivial problems, this process might require high computational resources (large memory and search times, for example), and thus, a variety of algorithmic issues are being studied to design efficient GAs. With this goal in mind, numerous advances are continuously being achieved by designing new operators [6], hybrid algorithms [7], and more. One of such improvements consists in using parallel models of GAs (PGAs) [3].

The class of multipopulation parallel GAs runs many elementary GAs working on separate subpopulations. Each subalgorithm includes an additional phase of periodic communication with a set of neighboring subalgorithms located on some topology (Figure 1). This communication usually consists in exchanging a set of individuals, although nothing prevents the subalgorithms from exchanging another kind of information such as population statistics. All the subalgorithms are thought to perform the same reproductive plan. Otherwise the PGA is said to be heterogeneous [3].

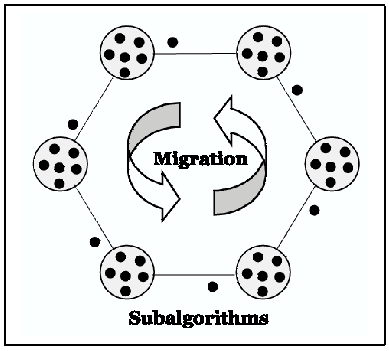

Figure 1: Parallel Genetic Algorithm.

4. THE REPULSION ALGORITHM

To enhance the numerical efficiency of other existing proposals we here develop a new local search algorithm for the ECC problem class: the Repulsion Algorithm (RA). This algorithm is based on Physics, concretely on Electrostatics.

The core idea comes from placing a set of particles charged with the same charge (and sign) in the surface of a sphere: they will tend to move away and get separated from each other. The forces produced by this configuration can be calculated by using the Coulomb Law. The words of a binary code can be seen as particles in an n-dimensional space, where n is the word length. One given solution for ECC is better than another when the minimum Hamming distance between codewords is greater in the first solution. So, the idea is to consider the words as particles, and then apply the Coulomb Law to calculate the forces among them; finally, we could simulate their free movement to reach a state of lower energy.

RA calculates the forces between codeword pairs (assuming they all have the same charge) and then it computes the resultant forces for all codewords. Let us call $\mathbf{f}_{ij}$ the force that codeword j exerts over codeword i, and let $\mathbf{F}_i$ be the resultant force applied over the codeword i by the contribution of all the particles (Figure 2).

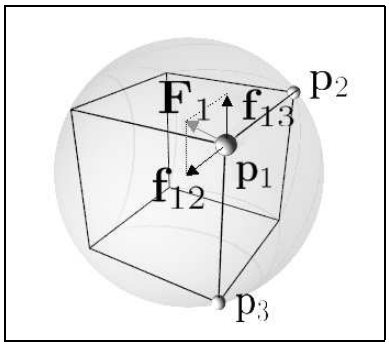

Figure 2: Forces among three particles and the resultant force for particle $\mathbf{p}_1$.

Once we know the resultant force of all codewords, we have to simulate their movement. As mentioned above, the codewords can only move to adjacent vertices. This means inverting a bit in the binary string. To establish what edge each codeword will move over, we break the resultant force vector down as the sum of two orthogonal vectors. One of

them is normal to the n-plane tangent to the n-sphere into which the n-cube is circunscribed (normal component), and the other is lain on the tangent n-plane (tangential component). The first one will be denoted as $\mathbf{F}_i^n$ and the second one is referred to as $\mathbf{F}_i^t$ (Figure 3).

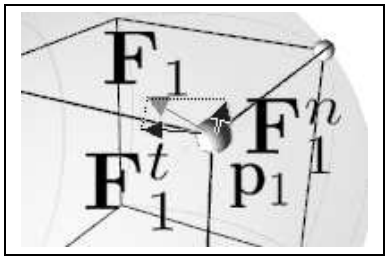

Figure 3: Normal and tangential components of the resultant force.

The vector orthogonal to the n-sphere is discarded because it contributes to the movement of the particle away from the n-sphere surface. So we focus on the tangential component ($\mathbf{F}_i^t$). If codewords could freely move over the surface of the n-sphere, the tangential component vector would represent the movement acceleration. Since codewords can only move through an edge, we need to determine the edge that forms the smaller angle with the tangential component, since it defines the movement leading to a faster decrease in energy. To do this, we assign a unitary vector to each edge. The vector origin is the vertex in which the particle is, and the target is the other extreme of the edge. There are as many edges as dimensions in a vertex, so we denote the edge vectors as $\mathbf{e}_i^k$, where i is the particle and k is the dimension (observe that the pair vertex-dimension determines an edge vector, but to make it easier, we use $\mathbf{e}_i^k$ to refer to a vector edge associated with the vertex $\mathbf{p}_i$). Then, we do the dot products $m_i^k = \mathbf{F}_i^t \cdot \mathbf{e}_i^k$ for each particle and dimension, and we choose for each particle i the dimension k for which m_i^k is maximum (minimum angle). This dimension determines the edge of movement, that is, the codeword will move through that edge.

When the tangential component of the force is too small this means that the particle is near an equilibrium position. In this equilibrium position, the particle do not move because it has reached a local minimum of the energy function (assuming that the other particles are motionless). For this reason, a minimum value or threshold τ is required for m_i^k in order to do the movement. From all the codewords suitable for movement, i.e., those with $m_i^k \geq \tau$, one is randomly chosen and "moved". The other particles remain unchanged. With this movement, an iteration of the algorithm is completed.

5. EMPIRICAL STUDY

In this section we discuss the representation of the solutions for the problem and the fitness function employed. Then, we describe the experiments and the parameters used. Finally, we show and comment the results obtained.

5.1 Representation and Fitness Function

To encode the solutions of the problem we use an $M \times n$ binary string. The genotype is the concatenation of the M codewords of length n. This representation directly maps the problem parameters into the chromosome of each individual, providing an easy conversion between the genotype and the phenotype for GAs. RA also finds this encoding natural.

As to the fitness function to maximize, one is tempted to use the minimum Hamming distance between the words in the code to measure the quality of a solution, but this is a too coarse function, since it gives a poor information for comparing two competing solutions. A more precise fitness function is that proposed by Dontas and De Jong [8], i.e.,

$$f(\mathbf{x}) = \frac{1}{\sum_{i=1}^{M} \sum_{j=1, j\neq i}^{M} \frac{1}{d_{ij}^2}} \quad (1)$$

where $\mathbf{x}$ is the code (codeword vector) and d_{ij} is the Hamming distance between x_i and x_j. This function measures how well the M words are placed in the corners of an n-dimensional space by considering the minimal energy configuration of M particles, as it is done in Physics. Although it has been successfully used in the past [2, 5, 8], this function has a subtle inconvenient: it can assign a higher fitness value to a code with a smaller minimum Hamming distance than other. That is, if C_1 and C_2 are two codes, it can occur that $f(C_1) < f(C_2)$ whereas $d(C_1) > d(C_2)$. To avoid this problem we have added a term to the equation that increases with respect to the minimum Hamming distance. The fitness function we have used is:

$$f'(\mathbf{x}) = \frac{1}{\sum_{i=1}^{M} \sum_{j=1, j\neq i}^{M} \frac{1}{d_{ij}^2}} + \left(\frac{d_{min}}{12} - \frac{d_{min}^2}{4} + \frac{d_{min}^3}{6} \right) \quad (2)$$

where d_{min} is the minimum Hamming distance between codewords in the code $\mathbf{x}$.

This function is broken down into two terms. The first one is just Equation (1), that allows a precise distinction between codes with the same minimum distance but different codewords arrangement. The second one guarantees that fitness values of codes with minimum distance d are smaller than those of codes with minimum distance $d+1$.

5.2 Algorithms and Parameterization

To solve the problem we use five algorithms: Repulsion Algorithm (RA), Genetic Algorithm (GA), Parallel Genetic Algorithm (PGA), a hybrid between a GA and the RA (GARA), and another hybrid, this time between a PGA and the RA (PGARA). Now we explain the parameters of each algorithm in detail.

The RA stops when it finds an optimum or reaches 10^5 iterations. The threshold τ is set to 0.001.

For the GA we use one single population of 480 individuals (panmixia). The algorithm uses binary tournament selection to select two individuals. These individuals are recombined using single point crossover with probability $p_c = 1.0$. The resultant individuals are mutated by using bit flip mutation with probability $p_m = 0.003$ (approximately the inverse of the string length). Finally, the individuals are inserted in the population only if they are better than the present worst ones. The stop criterion is to find an optimum or to reach 10^5 iterations.

We use three distributed GAs with 5, 10, and 15 islands to solve the problem. We call them PGA5, PGA10, and PGA15, respectively. The population size is the same in all three PGAs, that is, 480 individuals. These individuals are

equally distributed among all the islands. Selection, recombination, mutation, and replacement operators are the same as for the panmictic GAs. The subalgorithms are connected through a unidirectional ring and perform asynchronous migrations. Every eleven iterations one individual is selected from the population by using binary tournament and sent to the next subalgorithm in the ring. In the target subalgorithm, the individual is inserted in the population if it is better than the worst one in the population, replacing it. The stop criterion is to find the optimum or to reach 10^5 iterations. All these parameters are summarized in Table 1.

The GARA algorithm is a hybrid algorithm combining GA and RA. There are many ways of hibrydizing two algorithms. In this case one iteration of the RA is applied (one bit change) instead of the mutation in the GA loop. The parameters for GARA are the same as GA and RA.

The PGARA algorithm is similar to PGA but it uses RA as a mutation operator, as it is done in GARA. As in PGA, we use three PGARA algorithms with 5, 10, and 15 islands: PGARA5, PGARA10, and PGARA15. The parameters are summarized in Table 1.

	PGA5	PGA10	PGA15
Islands	5	10	15
Subpop. size	96	48	32
Selection	Bin. Tourn. (2 inds. selected)		
Recombination	Single Point ($p_c = 1.0$)		
Exploitation	If pure: Bit-Flip ($p_m = 0.003$)		
	If hybrid: RA 1 iter. ($\tau = 0.001$)		
Replacement	Elitist		
Topology	Unidirectional Ring		
Migration type	Asynchronous		
Migration gap	11		
Migr. Selection	Bin. Tourn. (1 ind. selected)		
Migr. Replacement	Local worst if incoming is better		
Stop criterion	Optimum or 10^5 iterations		

Table 1: Parameters of the parallel algorithms.

The machines used for the experiments are Pentium 4 at 2.4GHz with 512MB of RAM interconnected with a Fast Ethernet. For PGAn and PGARAn algorithms we performed the experiments on 1 CPU and 5 CPUs. We present average values of 30 independent runs.

In the parallel algorithms, the number of evaluations reported to find an optimum is the sum of all the evaluations made by the subalgorithms.

5.3 Experiments and Results

In this subsection we discuss the relative performance of the presented algorithms. Let us begin by analyzing the sequential algorithms (Table 2).

	%	Best fitness		Evaluations	
	Success	$\overline{\mathbf{x}}$	$\sigma_{\mathbf{n}}$	$\overline{\mathbf{x}}$	$\sigma_{\mathbf{n}}$
RA	00.00	3.16	1.53	–	–
GA	00.00	7.06	0.00	–	–
GARA	53.33	19.07	9.42	65771.00	35793.92

Table 2: Results of RA, GA and GARA: importance of the hybridization with RA.

It can be noticed that neither RA nor GA are able of reaching an optimum solution. However, the average best fitness obtained by GA is higher than for RA, so we can state that the GA works out solutions that are closer to the optimum than the RA ones. The hybrid algorithm GARA reaches an optimum solution in 53.33% of the cases, which is a clear indicator of its higher accuracy for this problem.

The results shown in Table 3 summarize the execution of PGA5, PGA10, and PGA15 on 5 CPUs.

	PGAn 5 CPUs				
	%	Best fitness		Evaluations	
n	Success	$\overline{\mathbf{x}}$	$\sigma_{\mathbf{n}}$	$\overline{\mathbf{x}}$	$\sigma_{\mathbf{n}}$
15	16.67	10.48	7.64	746532.00	603619.04
10	20.00	11.16	8.20	704941.00	438340.24
5	6.67	8.43	5.11	346982.00	62298.00

Table 3: Results of PGAn on 5 CPUs: parallel processes.

In this case, the parallel GA improves the accuracy obtained with sequential GA, that is, the structured population is beneficial for solving this problem, as often reported in literature [2]. Working with several subpopulations helps to maintain a high diversity, thus avoiding premature convergence. We observe this in the results, since PGA10 improves the success rate over PGA5. However, the results of PGA15 are similar to that of PGA10.

Now in Table 4 we present the results of the same PGAs executed on 1 CPU.

	PGAn 1 CPU				
	%	Best fitness		Evaluations	
n	Success	$\overline{\mathbf{x}}$	$\sigma_{\mathbf{n}}$	$\overline{\mathbf{x}}$	$\sigma_{\mathbf{n}}$
15	20.00	11.16	8.20	2144207.00	708207.40
10	20.00	11.16	8.20	656393.00	442050.00
5	13.33	9.80	6.97	265179.00	117525.02

Table 4: Results of PGAn on 1 CPU: concurrent processes.

As we expected, the numerical results are similar to that of using 5 CPUs. The only difference between 1 and 5 CPUs is the communication time, that is smaller in the first case. This fact accelerates the search and allows to get optimum solutions in 1 CPU while with 5 CPUs there was a smaller number of migrations per time unit. Again, we observe that using more than 10 islands does not improve the efficacy of parallel GA.

From the previous results, we can deduce that the utilization of hybrid and parallel algorithms separately improves the success percentage in the ECC problem. Therefore, the next idea is to use hybrid plus parallel techniques together, that is, parallel hybrid algorithms. The results for PGARAn on 5 CPUs are shown in Table 5.

	PGARAn 5 CPUs				
	%	Best fitness		Evaluations	
n	Success	$\overline{\mathbf{x}}$	$\sigma_{\mathbf{n}}$	$\overline{\mathbf{x}}$	$\sigma_{\mathbf{n}}$
15	93.33	26.47	4.24	84880.86	51874.80
10	80.00	23.73	7.78	116413.50	129890.60
5	43.33	16.22	10.03	72014.62	44079.04

Table 5: Results of PGARAn on 5 CPUs: parallel processes.

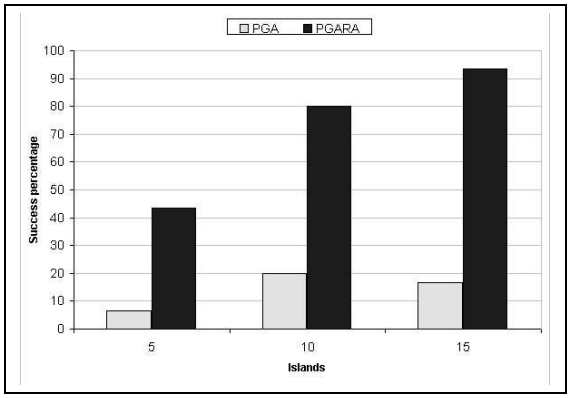

Figure 4: Results of all the parallel algorithms on 5 CPUs: importance of the hybridization with RA.

As we expected, the success percentage is higher than for pure PGAs, reaching 93.33% in PGARA15. The number of evaluations employed to find the solution is smaller than for any previous algorithm, except for GARA. Increasing the number of islands is, in this case, beneficial.

In Table 6 we show the results for PGARAn on 1 CPU.

	PGARAn 1 CPU				
	%	Best fitness		Evaluations	
n	Success	$\overline{x}$	σ_n	$\overline{x}$	σ_n
15	96.67	26.88	3.68	268485.32	534210.44
10	83.33	24.95	6.07	196861.60	377626.88
5	83.33	24.15	7.64	85541.20	64044.16

Table 6: Results of PGARAn on 1 CPU: concurrent processes.

It comes as no surprise that the success percentage is higher than for the 5 CPU version. In this table we find the best overall success percentage. PGARA15 shows a 96.67% success rate, thus indicating that 29 out of 30 runs got an optimum solution. The reason for the improvement in 1 CPU versus 5 CPUs comes from the reduced communication times that allow a slightly tighter liaison among the subalgorithms.

In Figure 4 we show the success percentage for all parallel algorithms run on 5 CPUs in a bar diagram. In this diagram we compare the results of PGA against PGARA, and it shows a neat advantage when RA hybridization is used.

The execution times of the runs do not exceed the eight minutes in any case. In some cases, e.g. PGARA10 in 1 CPU, the execution is performed in one second. The execution of RA takes eight seconds, what gives us an idea about the time cost of the algorithm.

The results reported in this work represent the best-known accuracy and efficiency outcomes for ECC. In [5], a SIMD architecture and a GSA algorithm are used to solve the problem. The algorithm requires more than 12000 parallel iterations with 256 processors (more than 3072000 evaluations), and more than 10000 with 16284 processors (more than 163840000 evaluations) to reach an optimum solution. This contrast with our 84880 evaluations with PGARA15 on 5 CPUs. In [2], we find competitive results, but the representation used for the problem is different and can not be used to find an optimum solution in all instances of ECC.

6. CONCLUSIONS AND FUTURE WORK

In this work we have developed a local search algorithm for the ECC problem based on the repulsion of particles over the surface of a sphere: the Repulsion Algorithm. To study its properties we have solved the problem with it, a Genetic Algorithm, and a hybrid algorithm between them. Our conclusions indicate that a great improvement in efficiency and accuracy can be got when the RA is used as local search, both for sequential panmictic (GARA) and distributed parallel algorithms (PGARA).

As a future work the algorithm will be studied in more detail. We can deal with other instances of the problem and it can be applied to solve other similar problems in which the analogy with Physics could be established. An example is the Thomson problem [9] consisting in placing charges in the surface of a sphere to get an equilibrium state.

7. ACKNOWLEDGEMENTS

This work has been partially funded by the Ministry of Science and Technology and FEDER under contracts TIC2002-04498-C05-02 (the TRACER project).

8. REFERENCES

[1] E. Agrell, A. Vardy, and K. Zeger. A Table of Upper Bounds for Binary Codes. *IEEE Transactions on Information Theory*, 47(7):3004–3006, 2001.

[2] E. Alba, C. Cotta, F. Chicano, and A. J. Nebro. Parallel Evolutionary Algorithms in Telecommunications: Two Case Studies. In *Proceedings of the CACIC02*, 2002.

[3] E. Alba and J. M. Troya. A Survey of Parallel Distributed Genetic Algorithms. *Complexity*, 4:31–52, 1999.

[4] T. Bäck, D. B. Fogel, and Z. Michalewicz. *Handbook of Evolutionary Computation*. Oxford University Press, New York NY, 1997.

[5] H. Chen, N. S. Flann, and D. W. Watson. Parallel Genetic Simulated Annealing: A Massively Parallel SIMD Algorithm. *IEEE Transactions on Parallel and Distributed Systems*, 9(2):126–136, 1998.

[6] C. Cotta, E. Alba, and J. M. Troya. Utilising Dynastically Optimal Forma Recombination in Hybrid Genetic Algorithms. In A. Eiben, T. Bäck, M. Schoenauer, and H. Schwefel, editors, *Parallel Problem Solving from Nature, PPSN V*, pages 305–314, Berlin, 1998. Springer-Verlag, Heidelberg.

[7] J. M. Daida, S. J. Ross, and B. C. Hannan. Biological Symbiosis as a Metaphor for Computational Hybridization. In L. Eshelman, editor, *Proceedings of the Sixth International Conference on Genetic Algorithms*, pages 328–335. Morgan Kaufmann, 1995.

[8] K. Dontas and K. De Jong. Discovery of Maximal Distance Codes Using Genetic Algorithms. In *Proceedings of the 2nd International IEEE Conference on Tools for Artificial Intelligence*, pages 905–811, Herndon, VA, 1990. IEEE Computer Society Press, Los Alamitos, CA.

[9] J. R. Morris, D. M. Deaven, and K. M. Ho. Genetic-Algorithm Energy Minimization for point charges on a sphere. *Physical Review B*, 53(4):1740–1743, 1996.