Empirical Studies on Parallel Network Construction of Bayesian Optimization Algorithms

Masaharu Munetomo
Information Initiative Center,
Hokkaido University
North 11, West 5
Sapporo, 060-0811 JAPAN.
munetomo@iic.hokudai.ac.jp

Naoya Murao
Graduate School of Engineering,
Hokkaido University
North 11, West 5
Sapporo, 060-0811 JAPAN.
naoya.m@cims.hokudai.ac.jp

Kiyoshi Akama
Information Initiative Center,
Hokkaido University
North 11, West 5
Sapporo, 060-0811 JAPAN.
akama@iic.hokudai.ac.jp

Abstract- This paper discusses a parallel optimization algorithm based on evolutionary algorithms with probabilistic model-building in order to design a robust search algorithm that can be applicable to a wide-spectrum of application problems effectively and reliably. Probabilistic model building genetic algorithm, which is also called estimation of distribution algorithm, is a promising approach in evolutionary computation and its parallelization has been investigated. We propose an improvement of parallel network construction in distributed Bayesian optimization algorithms which estimate distribution of promising solutions as Bayesian networks. Through numerical experiments on an actual parallel architecture, we show the effectiveness of our approach compared to the conventional parallelization. Also we perform experiments on a real-world application problem: protein structure predictions.

1 Introduction

No free lunch theorems for optimizations by Wolpert et. al.[WM97] show that the average performance of all optimization algorithms for all possible problems on discrete domain becomes the same. This indicates that we cannot have an all-round optimization algorithm effective to all possible problems, but does not deny the possibility to design a *robust* algorithm widely applicable and moderately effective to real-world application problems.

The objective of our research project is realizing robust optimization algorithms by developing advanced evolutionary algorithms although they need additional computational cost — the overheads should be resolved by developing algorithms suitable to parallel architectures. We seek for realizing a robust parallel optimization algorithm based on evolutionary algorithm with probabilistic model-building, which is also called estimation of distribution algorithm (EDA)[LL01].

Conventional evolutionary algorithms such as simple genetic algorithms (GAs) are vulnerable to loose linkage in encoding strings; their performances degrade when tight encoding is not preserved, that is, a set of loci related each other are not encoded tightly on a string, because classical genetic operators such as one-point crossover could easily disrupt promising subsolutions called *building blocks* (BBs). Tight encoding can only be ensured by using problem-specific knowledge given by the users. It is, however, sometimes difficult to obtain such knowledge, especially when the users try to solve challenging problems.

EDAs are designed to increase robustness of genetic search by introducing probabilistic model-building for a population of strings after selections. They build probabilistic models from distribution of alleles in a population of strings, which are employed to generate offsprings for the next generation. Genetic operators of conventional GAs are replaced by the probabilistic model-building and generation of offsprings based on the model. EDAs have been applied successfully to difficult problems where conventional GAs cannot solve due to their linkage sensitivity.

Bayesian optimization algorithm (BOA)[PGCP99] is considered one of the most sophisticated techniques among EDAs, which builds a probabilistic model as a Bayesian network based on distribution of alleles in a population of strings after selections. According to the network, BOA generates a population of strings for the next generation. Parallel implementations of the BOA have been investigated since its model-building process is costly. Distributed BOA (DBOA) proposed by Ocenasek[Jir02] parallelizes its model-building process and generation of offsprings based on the model. DBOA employs permutation of nodes in order to prevent cycles in its Bayesian network construction process.

In this paper, we propose an effective parallelization of BOA by detecting cycles periodically to be removed and controlling the number of edge additions between the detections in its Bayesian network construction process. We perform numerical experiments employing a trap test function to show the effectiveness of our approach compared with DBOA. We also perform an empirical study on a real-world application, protein structure prediction problems to demonstrate the effectiveness of our algorithm compared with other conventional approaches in parallel evolutionary algorithms.

0-7803-9363-5/05/$20.00 ©2005 IEEE.

2 Parallel Evolutionary Algorithms

In design processes, GAs are frequently employed to optimize design parameters because they only need input and output of the objective functions — a *black-box* optimization is possible by GAs and this is considered a primary advantage of evolutionary algorithms in general. Therefore, GAs have been applied to a wide-spectrum of application problems where conventional optimization algorithms were considered difficult to apply.

Since GAs need more computational cost than problem-specific algorithms but they are considered suitable for parallel computation, their parallel implementations are extensively investigated. Doctoral thesis by Cantú-Paz[CP99] investigates conventional approaches in parallel GAs (PGAs). In the thesis, the conventional approaches of PGAs such as master-slave and island models are analyzed theoretically and empirically.

Advanced methods for parallel evolutionary algorithms are also investigated such as parallel linkage identification techniques [MMA03, MM04] which identify a set of loci tightly linked in parallel to ensure effective genetic recombinations. Similar to EDAs, linkage identification increases robustness of search because it is less dependent upon whether tight linkage is ensured in encoding strings, and therefore it can be applied wider spectrum of application problems.

Parallel EDAs are another set of promising approaches for parallel evolutionary algorithm that parallelize their model-building process because majority of computational overheads of EDAs lie in the process. In the following, we review EDAs and their parallelized algorithms, especially the most sophisticated algorithm based on constructing Bayesian networks.

3 Estimation of Distribution Algorithms

In the field of evolutionary computations, estimation of distribution algorithms (EDAs) have been studied to design robust evolutionary search. Conventional GAs are not totally black-box because they need some *a priori* problem specific knowledge to ensure tight encoding. On the other hand, EDAs are not dependent upon tight encoding because they learn some structure of the target problem by estimating the probability distribution in a population of promising solutions.

EDAs build probabilistic models from a population of strings after selections and employ the model to create strings for the next generation. General EDAs perform the following sequence:

1. Randomly initialize a population of N strings.
2. Select M ($M < N$) strings with higher fitness (for maximization problems) from the population.
3. Create a probabilistic model base on the selected strings.
4. Create a population of N strings for the next generation based on the model.
5. If a specified termination condition is not satisfied, return to 2.

EDAs are population-based search algorithms similar to GAs, however, they do not employ explicit genetic operators such as crossovers and mutations. Instead, they estimate distributions of alleles for the selected strings and create strings for the next generations by employing a model of promising solutions.

The early EDAs such as PBIL (Population-Based Incremental Learning)[Bal94] and UMDA (Univariate Marginal Distribution Algorithm)[MBV96] estimate the probabilities of 1's occurrence in each locus and generates offsprings based on the probabilities. A series of advanced EDA methods have been proposed such as BMDA (Bivariate Marginal Distribution Algorithm)[PM98], FDA (Factorial Distribution Algorithm)[MM99], and BOA (Bayesian Optimization Algorithm)[PGCP99], which consider conditional probabilities among loci.

3.1 Bayesian optimization algorithm

The Bayesian optimization algorithm (BOA) proposed by Pelikan et. al[PGCP99] is considered one of the most sophisticated algorithm in EDAs. BOA builds probabilistic models based on Bayesian networks to represent joint probabilities of alleles for pairs of loci. In the Bayesian networks, a node i ($i = 0, \cdots, l-1$, where l is the string length) represents a random variable X_i, the probability of 1's occurrence on locus s_i, and a directed link from node i to node j represents node i is a parent of node j. $p(X_i|\prod_{X_i})$ is the probability distribution of X_i where $\prod_{X_i}$ is the set of parents (set of nodes connected to node i) of X_i. This is illustrated in Figure 1.

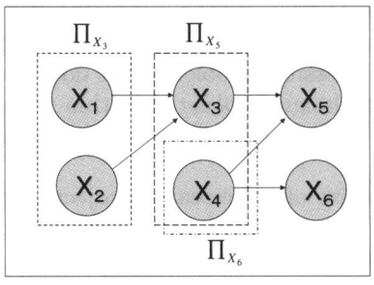

Figure 1: An example of Bayesian network.

The following is the algorithm of the original BOA[PGCP99]:

1. Initialize population $P(0)$ randomly and set $t = 0$
2. Select a set of promising strings $S(t)$ from $P(t)$
3. Construct a Bayesian network B based on the BD metric

4. Generate a set of offsprings $O(t)$ based on the joint probabilities specified by the network B

5. Create a population $P(t+1)$ for the next generation by replacing strings from $P(t)$ with $O(t)$ and set $t = t + 1$

6. If a termination criterion is not satisfied, return to 2.

Similar to GAs, BOA starts with a population of randomly initialized strings and applies selections to obtain a set of promising strings with relatively higher (in maximization problems) fitness values. Based on the selected strings, it generates a Bayesian network B that minimizes a metric to measure the quality of the network. The original BOA employs the Bayesian Dirichlet (BD) metric [HC94] $p(D, B|\xi)$ defined as follows:

$$p(D, B|\xi) = p(B|\xi) \prod_{i=0}^{n-1} \prod_{\pi_{X_i}} \frac{m'(\pi_{X_i})!}{(m'(\pi_{X_i}) + m(\pi_{X_i}))!} \times \prod_{x_i} \frac{(m'(x_i, \pi_{X_i}) + m(x_i, \pi_{X_i}))!}{m'(x_i, \pi_{X_i})!} \quad (1)$$

where $p(B|\xi)$ is a prior probability of network B, $m(\pi_{X_i})$ represents the number of data where $\prod_{X_i} = \pi_{X_i}$ in D, $m(x_i, \pi_{X_i})$ is the number of data where $X_i = x_i$ and $\prod_{X_i} = \pi_{X_i}$ in D (π_{X_i} is the set of parental nodes of X_i and $m(\pi_{X_i}) = \sum_{x_i} m(x_i, \pi_{X_i})$). $m'(x_i, \pi_{X_i})$ is obtained with $p(B|\xi)$ from prior information of the network $m(x_i, \pi_{X_i})$ concerning the given problem.

As for the prior probability, $p(B|\xi)$, $p(B|\xi) = c\kappa^{\delta}$ is employed according to Heckerman[HC94], where c is a constant for normalization, $\kappa \in (0, 1]$ is a constant, and δ is the number of edges which differs between network B and that for the prior probabilities.

In order to control computational cost of the network constructions, maximum degree of the network is constrained to k. BOA performs a local search to minimize the metric under this constraint. The search process is as follows[PGCP99]:

1. Initialize a network consisting only of nodes without links.

2. A pair of nodes (u, v) is randomly selected from nodes.

3. If a directed edge $e = (u, v)$ already exists in the current network or it should cause a cycle, go to 6.

4. Calculate the BD metric for the network adding the edge e to the current one.

5. If the metric has improved, the edge e is added to the current network. Otherwise, it is not added.

6. Unless a termination condition is satisfied, return to 2.

To generate new strings for the next generation, the joint probability

$$p(X) = \prod_{i=0}^{l-1} p(X_i | \prod_{X_i}) \quad (2)$$

is employed based on the constructed Bayesian network.

As for computational complexity of the BOA, its network construction — searching an optimal network that minimizes the BD metric — tends to dominate its computational overhead. Computational complexity for searching network structure of BOA is $O(k2^k l^2 N + kl^3)$ where k is the maximum degree (maximum number of links connected to a node) and N is the number of strings[PGCP99].

3.2 Distributed BOA

As stated above, majority of computational overheads of the BOA lies in its model building process to generate Bayesian networks that minimize some metric, which should be parallelized. In order to parallelize the network construction, distributed BOA (DBOA) by Ocenasek[Jir02] assigns each node of the network to each processor and calculates the metric independently. Since Bayesian networks should not have cycles, DBOA randomly generates a permutation of nodes which indicates priorities among them in connecting directed edges. For example, a permutation of nodes (4 1 3 5 2) indicates that edges can only be connected from node 4 to nodes { 1, 3, 5, 2 }, and from node 1 to { 3, 5, 2 }, and so on. This is illustrated in figure 3.2.

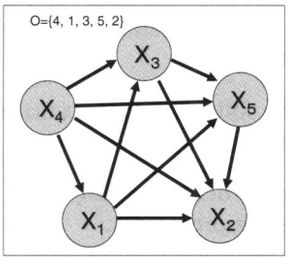

Figure 2: Nodes and edges that can be added according to a permutation (4 1 3 5 2).

The permutation is broadcast to all processors, which restricts network construction in each processor in order not to generate cycles. The permutation is randomly regenerated periodically along the search process, because such restrictions may cause unsearched regions in search of networks.

The following shows the algorithm of the parallel Bayesian network construction process in the DBOA.

1. A master processor randomly initializes an initial network consisting only nodes without edges and broadcasts the current population of strings to all slave processors.
2. Repeats the following until a termination condition is satisfied.
3. The master randomly initializes a permutation of nodes and broadcast it and current network to all slave processors.
4. Each slave processor performs the following (a)–(e) on their assigned subset of nodes.
   (a) For each node i in the subset, perform the following (b)–(e).
   (b) A node j is selected from the nodes located after i in the permutation.
   (c) Calculate the difference of metric by adding an edge from i to j.
   (d) Find j^* that maximizes the improvement of the metric by repeating (b)–(c).
   (e) Add an edge from i to j^* to the network.
5. The master collects networks generated in slave processors to update its network.

Performance of the parallel network construction is dependent upon randomly generated permutation of nodes. DBOA tries to avoid bias caused by the permutation by re-initializing it periodically, however, even though such re-initialization is employed, restricting search space with the permutation may derail search paths to the optimal network, which leads to convergence to local optima.

Overall algorithm of DBOA performs the following:

1. A master processor randomly initializes a population of strings.
2. Repeats the following 3.–4. until a termination condition is satisfied.
3. Parallel construction of Bayesian networks assigned to slave processors.
4. Parallel generation of offsprings assigned to slave processors based on the obtained network.

Aside from the parallel network construction, generation of offsprings can also be parallelized easily because the process of string generations based on the probability distribution with the Bayesian network is highly independent each other. Therefore, DBOA parallelizes generations of offsprings in each slave processor and collects them to the master processor which broadcasts them to the slaves for parallel network construction in the next generation.

4 Parallel Bayesian network construction

As discussed in the previous section, the conventional approach for parallel Bayesian network construction in DBOA may cause unreliable results because it strongly depends upon the permutations restricting search space to avoid cycles in the parallel network construction process.

We seek for another approach of the parallel network construction without employing such restrictions to improve the quality of the network. Our approach is based on random addition of edges in parallel with some roll-back mechanism when it causes cycles.

Since decisions on edge additions are performed in each processor independently, just adding edges in parallel may cause cycles if no information is exchanged among processors. Therefore, each processor should exchange information to check whether a cycle exists in the overall Bayesian network by merging the networks in slave processors. When a cycle detected in the merged network, we invoke a roll-back process that removes the cycle from the network.

On the other hand, frequent exchange of information among processors is unrealistic because it causes overheads in communications which degrade overall performances considerably. Therefore, we need to reduce communication overheads as much as possible, and at the same time, we need to remove cycles effectively and reliably. The tradeoff between the communication overheads and accuracy of cycle detection is resolved by controlling frequency of communications.

In our approach, parallel network construction is divided into *stages*, which are repeatedly performed in each processor. In each stage, each processor repeats its assigned edge additions independently, which is followed by broadcasting the results to check cycles. When we detect a cycle in the network, we remove an edge randomly selected from it.

We control the number of edges to be added in each stage to control the frequency of communications. We set the number of edge additions T_i of the i-th stage as follows:

$$T_i = \begin{cases} a \times \frac{l}{P} & \text{if } i = 0 \\ c \times T_{i-1} & \text{otherwise,} \end{cases} \tag{3}$$

where l is the strings' length, P is the number of processors, and $a > 0$, $0 < c < 1$ are constants to control T_i. The value of T_i decreases along the stages because adding edges to the network should increase the possibility to cause cycles; we need to check them more frequently.

The parallel network construction algorithm we propose performs the following sequence in parallel processors:

1. Each processor generates an initial network consisting only of nodes without edges, sets $i = 0$, and calculates T_0.
2. Each processor performs the following (a)–(b) T_i

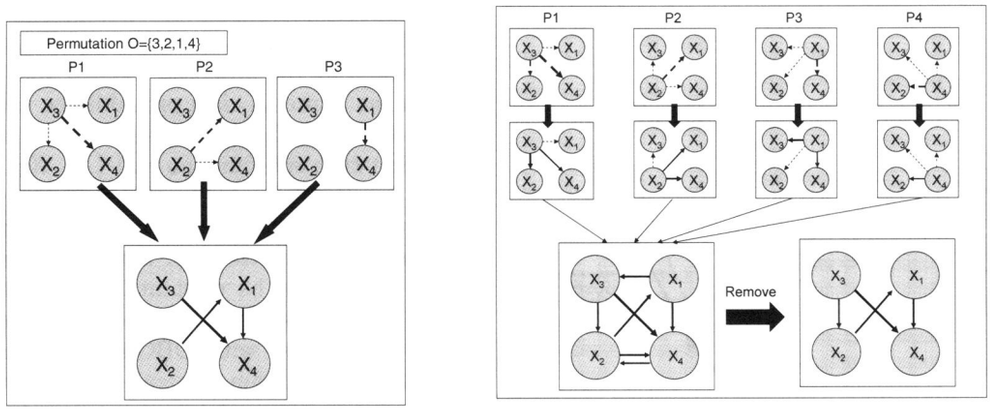

Figure 3: A comparison of DBOA (left) and our approach (right) in parallel network construction.

times as one stage for its assigned nodes.

(a) It calculates a difference of metric by adding an edge from a node selected from its assigned nodes to one selected from all nodes.

(b) By calculating the above for all possible edges, select an edge that maximizes the improvement of the metric to be added to the current network.

3. Each processor broadcasts its added edges in the stage.

4. Each processor collects the added edges from the other processors and updates its current network structure.

5. For any cycle existent in the current network, an edge is randomly selected from the cycle to be deleted from the network. This is performed in one selected processor to avoid inconsistencies of the network, and the resultant network is broadcast to other processors.

6. If a termination condition is not satisfied, we set $i \leftarrow i+1$, update T_i, and return to 2.

Figure 3 illustrates the difference of the parallel network construction between our approach and the conventional method in the DBOA.

Conventional approach in DBOA restricts search space by introducing permutations of nodes in order not to generate any cycles in a network, which may create unsearched region in the search space of the network. On the other hand, our approach does not employ such restriction and allows cycles to be generated in each processor, which should be removed by checking them periodically.

5 Numerical Experiments

We perform experiments on a parallel computer to show the effectiveness of our algorithm in comparison with the conventional DBOA. For the following experiments, we employ a parallel computer SGI Onyx 300 consisting of MIPS R14000/600MHz × 32 CPUs with 16GB shared memory connected via NUMAflex^{TM} high-speed network. Message Passing Interface (MPI) is used for communications among processors.

We use the following test function consisting of *trap* sub-functions defined on a binary string s usually employed to test evolutionary algorithms.

$$f(s) = \sum_{i=0}^{l/k} \mathrm{trap}_k(u_i), \tag{4}$$

$$\mathrm{trap}_k(u_i) = \begin{cases} k - u_i - 1 & \text{if } 0 \le u_i \le k-1 \\ k & \text{if } u_i = k, \end{cases} \tag{5}$$

where l is the length of the string s, k is the length of the sub-string s_i of s, and u_i is the number of 1's occurrence in s_i.

When each substring of each sub-function is tightly encoded on a string, conventional GAs can find its optimal solutions. It becomes difficult when such tight encoding is not ensured for the sub-functions. When tight encoding is not ensured, we need to employ advanced methods such as linkage identifications and EDAs. Performances of such advanced methods are not dependent upon tight linkage, that is, results should be the same for problems whether tight encoding is ensured or not. This is the reason why such advanced algorithms are called *robust* optimization algorithms.

Figure 4 shows comparisons of overall execution time by employing 8 processors to obtain optimal solutions of the 5-bit trap test function ($k = 5$ in equation(4)) with our approach and DBOA. In the figure, the x-axis shows string length l and the y-axis is the overall execution time to obtain optimal solutions. As the length increases, time to obtain solutions increases rapidly because the size of the search space is $O(2^l)$. The proposed algorithm can obtain solutions with less computational cost. When $l = 500$, our algorithm improves more than 20 % of execution time compared with DBOA.

Figure 5 shows speedup factors $S = T_s/T_p$ (T_s: time by serial processing, T_p: time by parallel processing)

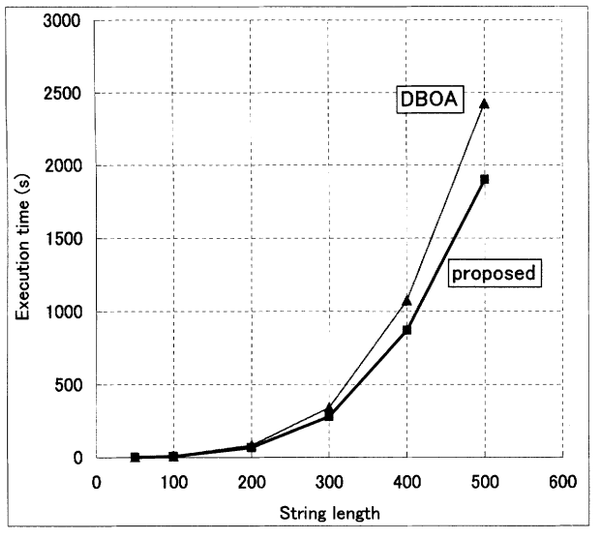

Figure 4: Comparison of execution time vs. string length.

for the test functions with $l = 105$ and $k = 5$. These results show that the proposed algorithm shows higher speedups compared to DBOA, which shows around 10 % improvement. We expect more improvement of the speedup as the number of processors increases.

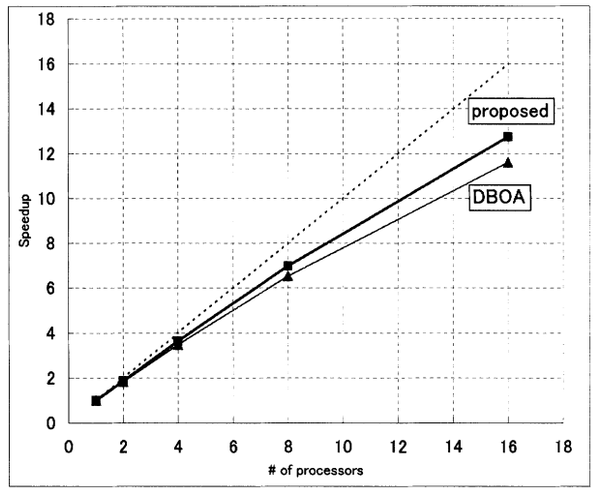

Figure 5: Comparison of speedup factors for 5-bit trap functions ($k = 5$).

To show where such improvement comes from, we compare the average number of generations for both approaches. By the proposed algorithm, the average becomes 25.3, while conventional DBOA needs 38.7 generations.

Such improvement on the number of generations comes from improvement of accuracy in network constructions by removing strict restrictions in DBOA based on a fixed permutation of nodes. On the other hand, in our method, by introducing random additions of edges and deletion of those in cycles, overall overheads in our approach become less than those by DBOA. Figure 6 shows a part of Bayesian networks generated by the proposed method and the DBOA.

In the figure, X_i is a probabilistic variable for a locus, and $\{X_0, X_1, X_2, X_3, X_4\}$ is a set of variables for the first trap sub-function and $\{X_5, X_6, X_7, X_8, X_9\}$ is that for the second. In the Bayesian networks correctly generated for the problem, no link should be generated among the sets. Dotted line represents links that should not be existent from the problem structure. The proposed method generates less invalid links than those with DBOA. Apparently, this indicates our algorithm generates more accurate results in the early stage of learning networks. In the 9-th generation, our algorithm converges on the first 5 bits ($\{X_0, X_1, X_2, X_3, X_4\}$), which results in no links are necessary in the network of conditional probabilities.

Table 1: Comparisons of the percentages of correct edges added.

Generation	1st	3rd	6th	9th
Proposed	30.1%	53.0%	72.2%	74.5%
DBOA	24.5%	44.3%	54.9%	66.5%

Table 1 shows the percentages correct edges added as a Bayesian network for the proposed algorithm and DBOA. The table shows that our approach achieves more accurate results from the early generations in the model-building process compared with DBOA, which leads to faster convergence to optimal solutions.

6 An Example of Applications to Real-world Problems: Protein Structure Prediction

In addition to the results on test functions, we should consider the efficiency of the algorithm from practical point of view comparing conventional genetic algorithms and their hybrids. We are applying the proposed algorithm to a real-world application problem — a molecular structure prediction problem that minimizes structural energy of complex molecules. In this section, we show a part of current results that indicates effectiveness compared with conventional approaches by a genetic algorithm and its hybrid with simulated annealing.

In the protein structure prediction problem, given sequence information of a protein, we determine its 3-dimensional structure by calculating a list of dihedral axes of elements that minimizes its structural energy.

Here, we try to solve three instances — Met-enkephalin, C-peptide, and Protein-G as shown in table 2 by employing our algorithm and conventional approaches such as Dual DGA[THT00] — a distributed GA model, and PSA/GAc[TYO02] — parallel simulated annealing with genetic operators.

We encode each dihedral axes into a 12-bit fixed-point value that ranges $[0, 2\pi)$. Therefore, string length

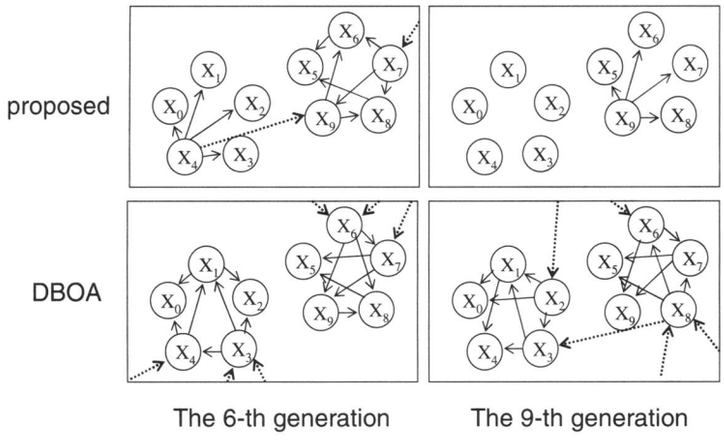

Figure 6: Bayesian networks generated by the proposed algorithm and the DBOA.

Table 2: Comparison of Obtained Structural Energy by the proposed and the conventional algorithms: *Dual DGA and **PSA/GAc.

			Energy (kcal/mol)		
Proteins	# of amino-acid residue	# of dihedral axes	Proposed	Conventional	Optimal
Met-enkephalin	5	19	-12.229	-12.188*	$\leq$ -11
C-peptide	13	64	-45.127	-33.721*	$\leq$ -42
Protein-G	56	275	-428.694	-356.790**	NA

for Met-enkephalin is 60, that for C-peptide is 156, and that for Protein-G is 672. "Optimal" in the last row indicates an empirically estimated optimal region of the structural energy. We employ 4 processors of the Onyx 300 for the experiments. To calculate the structural energies, we employ ECEPP/3[EH97].

For Met-enkephalin, both our algorithm and Dual DGA can obtain optimal results. On the other hand, for C-peptide, Dual DGA cannot reach to the optimal region, while our method succeeds in obtaining optimal solutions. Concerning computational cost, our algorithm needs less fitness evaluations than that with Dual DGA. The proposed algorithm needs 300,000 fitness evaluations for the Met-enkephalin and 700,000 for the C-peptide. On the other hand, Dual DGA needs 1,900,000 evaluations for the Met-enkephalin. For the most difficult Protein-G, our algorithm achieves better result than that of Protein-G, although both algorithms cannot obtain optimal structure empirically observed by X-ray analysis.

Figure 7 and 8 illustrate the obtained structures by our algorithm for Met-enkephalin and C-peptide, respectively. The obtained result empirically matches the observed structure of the actual proteins.

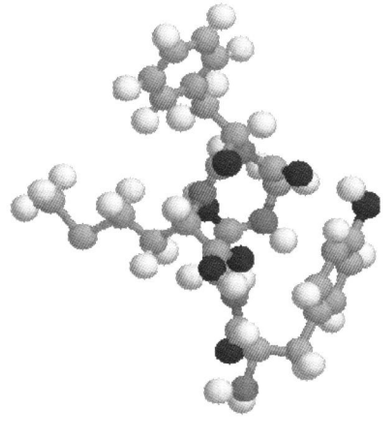

Figure 7: Structure of the Met-enkephalin optimized by the proposed algorithm.

7 Conclusion

Realizing a robust parallel search is a key to success in automating design process because high performance computing environments enables such an algorithm to replace a part of human trial and error processes in design optimizations. In this paper, we removed the restriction in conventional parallel network constructions in BOA and showed the effectiveness of our approach through experiments on actual parallel processors.

For the molecular structure identification problems that minimize structural energy of complex molecules, our current results indicate the possibility of having more accurate structures than those by conventional approaches, especially for larger problem sizes.

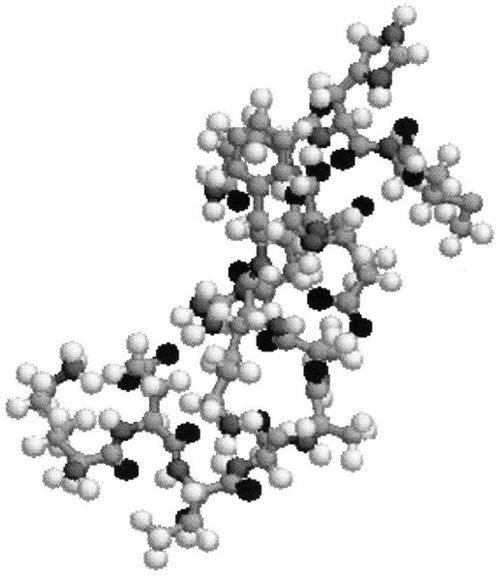

Figure 8: Structure for the C-peptide optimized by the proposed algorithm.

Bibliography

[Bal94] S. Baluja. Population-based incremental learning: A method for integrating genetic search based function optimization and competitive learning. Tech. Rep. No. CMU-CS-94-163, Carnegie Mellon University, Pittsburgh, PA, 1994.

[CP99] Erick Cantú-Paz. *Designing Efficient and Accurate Parallel Genetic Algorithms*. PhD thesis, University of Illinois at Urbana-Champaign, 1999.

[EH97] Frank Eisenmenger and Ulrich H.E. Hansmann. Global minimum configuration of a small peptide for the ECEPP/2 and ECEPP/3 force field. *Chemical Physics Letters*, (286):86–92, 1997.

[HC94] D. Heckerman and M. Chickering. Learning bayesian networks: The combination of knowledge and statistical data. Technical Report Technical Report MSR-TR-94-09, Microsoft Research, 1994.

[Jir02] Ocenasek Jiri. *Parallel Estimation of Distribution Algorithms*. PhD thesis, Brno University of Technology, 2002.

[LL01] Pedro Larrañaga and Jose A. Lozano. *Estimation of Distribution Algorithms: A New Tool for Evolutionary Computation*. Kluwer Academic Publishers, 2001.

[MBV96] H. Mühlenbein, J. Bendisch, and H.-M. Voigt. From recombination of genes to the estimation of distributions I. Binary parameters. *Parallel Problem Solving from Nature IV*, pages 188–197, 1996.

[MM99] Heinz Mühlenbein and Thilo Mahnig. FDA – a scalable evolutionary algorithm for the optimization of additively decomposed functions. *Evolutionary Computation*, 7(4):353–376, 1999.

[MM04] Kiyoshi Akama Masaharu Munetomo, Naoya Murao. Empirical investigations on parallelized linkage identification. In et al. Xin Yao, editor, *Parallel Problem Solving from Nature – PPSN VIII*, number 3242 in Lecture Notes in Computer Science, pages 322–331, 9 2004.

[MMA03] 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.

[PGCP99] M. Pelikan, D. E. Goldberg, and E. Cantú-Paz. BOA: The bayesian optimization algorithm. In *Proceedings of the Genetic and Evolutionary Computation Conference 1999 (GECCO-99)*, pages 525–532. Morgan Kaufmann Publishers, 1999.

[PM98] Martin Pelikan and Heinz Mühlenbein. Marginal distribution in evolutionary algorithms. In *Proceedings of the International Conference on Genetic Algorithms Mendel '98*, pages 90–95, Brno, Czech Republic, 1998.

[THT00] Masahiro Hamasaki Tomoyuki Hiroyasu, Mitsunori Miki and Yusuke Tanimura. A new model of parallel distributed genetic algorithms for cluster systems: Dual individual dgas. *High Performance Computing, Lecture Notes in Computer Sicence 1940*, pages 374–383, 2000.

[TYO02] Mitsunori Miki Maki Ogura Takeshi Yoshida, Tomoyuki Hiroyasu and Yuko Okamoto. Energy minimization of protein tertially structure by parallel simulated anneling using genetic crossover. In *Workshop of the Biological Application of Genetic and Evolutionary Computation, GECCO2002*, New York, USA, 2002.

[WM97] David H. Wolpert and William G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, 1997.