Design of Multithreaded Estimation of Distribution Algorithms

Jiri Ocenasek[1], Josef Schwarz[2], and Martin Pelikan[1]

[1] Computational Laboratory (CoLab), Swiss Federal Institute of Technology ETH, Hirschengraben 84, 8092 Zürich, Switzerland
jirio@inf.ethz.ch, pelikanm@inf.ethz.ch
[2] Faculty of Information Technology, Brno University of Technology, Bozetechova 2, 612 66 Brno, Czech Republic
schwarz@fit.vutbr.cz

Abstract. Estimation of Distribution Algorithms (EDAs) use a probabilistic model of promising solutions found so far to obtain new candidate solutions of an optimization problem. This paper focuses on the design of parallel EDAs. More specifically, the paper describes a method for parallel construction of Bayesian networks with local structures in form of decision trees in the Mixed Bayesian Optimization Algorithm. The proposed Multithreaded Mixed Bayesian Optimization Algorithm (MMBOA) is intended for implementation on a cluster of workstations that communicate by Message Passing Interface (MPI). Communication latencies between workstations are eliminated by multithreaded processing, so in each workstation the high-priority model-building thread, which is communication demanding, can be overlapped by low-priority model sampling thread when necessary. High performance of MMBOA is verified via simulation in TRANSIM tool.

1 Introduction

Estimation of Distribution Algorithms (EDAs) [1], also called Probabilistic Model-Building Genetic Algorithms (PBMGAs) [2] and Iterated Density Estimation Evolutionary Algorithms (IDEAs) [3], have been recently proposed as a new evolutionary technique to allow effective processing of information represented by sets of high-quality solutions. EDAs incorporate methods for automated learning of linkage between genes of the encoded solutions and incorporate this linkage into a graphical probabilistic model. The process of sampling new individuals from a probabilistic model respects these mutual dependencies, so that the combinations of values of decision variables remain correlated in high-quality solutions. The quality of new offspring is affected neither by the ordering of genes in the chromosome nor the fitness function epistasis. It is beyond the scope of this paper to give a thorough introduction to EDAs, for a survey see [2],[1].

Recent work on parallel Estimation of Distribution Algorithms concentrated on the parallelization of the construction of a graphical probabilistic model, mainly on the learning of structure of a Bayesian network. The Parallel Bayesian

Optimization Algorithm (PBOA) [4] was designed for pipelined parallel architecture, the Distributed Bayesian Optimization Algorithm (DBOA) [5] was designed for a cluster of workstations and the Parallel Estimation of Bayesian Network Algorithm (Parallel EBNA_{BIC}) [6] was designed for MIMD architecture with shared memory, also called PRAM.

Exponential size of the tabular representation of local conditional probability distributions emerged as a major problem in learning Bayesian networks. The most advanced implementations of EDAs use decision trees to capture local distributions of a Bayesian network more effectively. Nevertheless, parallelization of EDAs using Bayesian networks with decision trees has not been tackled yet. This paper—although applicable to parallel Bayesian network learning in general—deals especially with the parallel learning of decision trees and proposes advanced multithreaded techniques to accomplish it effectively.

The following section introduces the principles of Estimation of Distribution Algorithms. We identify the disadvantages of Bayesian network with tabular representation of local structures and motivate the usage of decision trees. Section 3 provides an overview of present parallel Estimation of Distribution Algorithms. We identify the main differences between them, analyze the way these algorithms learn dependency graphs and derive design guidelines. In Section 4 a new Multithreaded Mixed Bayesian Optimization Algorithm is proposed. We develop an original technique for parallel construction of Bayesian networks with decision trees using a cluster of workstations. Additional efficiency is achieved on a multithreaded platform, where the communication latencies between workstations are overlapped by switching between model-building thread and model-sampling thread. The efficiency and scalability of the proposed algorithm is verified via simulation using TRANSIM tool and demonstrated in Section 5.

2 Estimation of Distribution Algorithms

2.1 Basic principles

The general procedure of EDAs is similar to that of genetic algorithms (GAs), but the traditional recombination operators are replaced by (1) estimating the probability distribution of selected solutions and (2) sampling new points according to this estimate:

```
Set t := 0;
Randomly generate initial population P(0);
while termination criteria are not satisfied do begin
  Select a set of promising parent solutions D(t) from P(t);
  Estimate the probability distribution of the selected set D(t);
  Generate a set of new strings O(t) according to the estimate;
  Create a new pop. P(t+1) by replacing part of P(t) by O(t);
  Set t := t+1;
end
```

2.2 EDAs with Bayesian network model

First EDAs differed mainly in the complexity of the used probabilistic models. At the present time Bayesian networks have become most popular for representing discrete distributions in EDAs because of their generality. For the domain of possible chromosomes $\mathbf{X} = (X_0, \ldots, X_{n-1})$ the Bayesian network represents a joint probability distribution over $\mathbf{X}$. This representation consists of two parts—a dependency graph and a set of local conditional probability distributions.

The first part, the dependency graph, encodes the dependencies between values of genes throughout the population of chromosomes. Each gene corresponds to one node in the graph. If the probability of the value of a certain gene X_i is affected by a value of another gene X_j, then we say that "X_i depends on X_j" or "X_j is a parent variable of X_i". This assertion is expressed by the existence of an edge (j, i) in the dependency graph. A set of all parent variables of X_i, denoted $\mathbf{\Pi_i}$, corresponds to the set of all starting nodes of edges ending in X_i. The second part, the set of local conditional probability distributions $p(X_i|\mathbf{\Pi_i})$, is usually expressed in the tabular form.

The well known EDA implementations with the Bayesian network model are the Bayesian Optimization Algorithm (BOA) [7], the Estimation of Bayesian Network Algorithm (EBNA) [8] or the Learning Factorized Distribution Algorithm (LFDA) [9].

2.3 EDAs with decision tree models

For each possible assignment of values to the parents $\mathbf{\Pi_i}$, we need to specify a distribution over the values X_i can take. In its most naive form, local conditional probability distributions $p(X_i|\mathbf{\Pi_i})$ are encoded using tabular representation that is exponential in the number of parents of a variable X_i. More effective representation can be obtained by encoding local probability distributions using n decision trees. Since decision trees usually require less parameters, frequency estimation is more robust for given population size N.

The implementation of the Bayesian Optimization Algorithm with decision trees appeared in [10] and an extension of this idea for continuous variables resulted in the Mixed Bayesian Optimization Algorithm (MBOA) [11]. See also [12] for the multiobjective version of MBOA with epsilon-archiving approach.

3 Parallel Estimation of Distribution Algorithms

3.1 Performance issues

The analysis of BOA complexity identifies two potential performance bottlenecks — the construction of a probabilistic model and the evaluation of solutions. It depends on the nature of final application which part of MMBOA should be parallelized.

For problems with computationally cheap fitness evaluation—such as spin glass optimization with complexity $\mathrm{O}(n * N)$—great speedup can be achieved by parallelizing model construction, which can take $O(n^2 * N * log(N))$ steps in each generation. On the other hand, for problems with expensive fitness evaluation—such as ground water management optimization [13]—fitness evaluation should be also parallelized to achieve maximum efficiency.

In this paper we focus only on the parallelization of model building because it is not straightforward and needs special effort to be done effectively whereas the parallel evaluation of solutions can be done in obvious way and you can find many papers on this topic. For further analysis of parallelization of fitness evaluation see Cantú-Paz [14].

3.2 Parallel learning of dependency graph

All present parallel EDAs use greedy algorithms driven by statistical metrics to obtain the dependency graph. The so-called BIC score in EBNA [8] algorithm and also the Bayesian-Dirichlet metric in PBOA [4] and DBOA [5] algorithms are decomposable and can be written as a product of n factors, where the i-th factor expresses the influence of edges ending in the variable X_i. Thus, for each variable X_i the metrics gain for the set of parent variables $\mathbf{\Pi_i}$ can be computed independently. It is possible to utilize up to n processors, each processor corresponds to one variable X_i and it examines only edges leading to this variable.

The addition of edges is parallel, so an additional mechanism has to be used to keep the dependency graph acyclic. Parallel EBNA uses master-slave architecture, where the addition of edges is synchronous and controlled by the master.

PBOA and DBOA use predefined topological ordering of nodes to avoid cycles in advance. At the beginning of each generation, a random permutation of numbers $\{0, 1, ..., n-1\}$ is created and stored in an array $\mathbf{o} = (o_0, o_1, ..., o_{n-1})$. The direction of all edges in the network should be consistent with the ordering, so the addition of an edge from X_{o_j} to X_{o_i} is allowed if only $j < i$. The advantage is that each processor can create its part of the probabilistic model asynchronously and independently of other processors. No communication among processors is necessary because acyclicity is implicitly satisfied. The network causality might be violated by this additional assumption, but according to our empirical experience the quality of generated network is comparable to the quality of sequentially constructed network (see [4]).

3.3 Parallel offspring generation

The difference between various parallel EDAs arises more if we analyze how the offspring is generated.

PBOA was proposed for fine-grained type of parallelism with tightly connected communication channels. It can generate offspring in a linear pipeline way, because in the fine-grained architecture there are negligible communication latencies. It takes n cycles to generate the whole chromosome, but $n + N$ cycles to generate the whole population. For example let us consider the generation of first offspring chromosome. Its o_0-th bit is generated independently in processor number 0 at time t. Then, its o_1-th bit is generated in processor number 1 at time

$t+1$ conditionally on the o_0-th bit received from neighbouring processor number 0, etc. Generally, X_{o_i} is generated in CPU number i at time $t+i$ conditionally on $\mathbf{\Pi_{o_i}} \subseteq \{X_{o_0}, ..., X_{o_{i-1}}\}$. The advantage is that each conditional probability distribution is sampled locally at the place where it has been estimated.

In DBOA case communication delays are too long to use pipelined generation of offspring. Thus, DBOA uses distributed generation of offspring, each processor generates one complete subpopulation of chromosomes. See Fig. 1 for comparison of both types of offspring generation. Note that for this kind of offspring generation each processor needs to use a complete probabilistic model, which is constructed piecewise. Thus, gathering of local parts of model is a necessary step between model estimation and offspring generation, when each processor exchanges its part of model with the other processors.

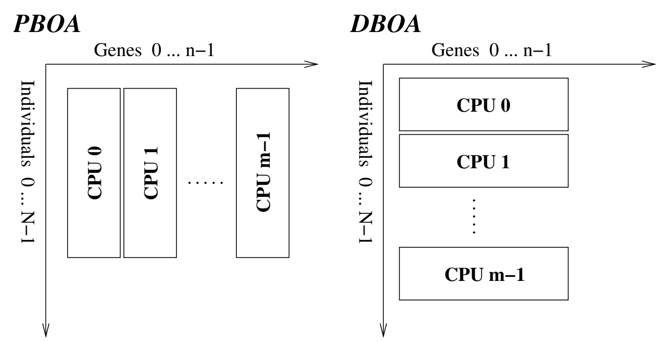

Fig. 1. Comparison of offspring generation. PBOA distributes the work between processors "horizontally" and DBOA "vertically".

In the case of parallel EBNA the target architecture is aimed to be the shared-memory MIMD, so the problem with the distribution of new population does not occur, the shared memory architecture allows for both "horizontal" as well as for "vertical" layout.

4 Multithreaded Mixed Bayesian Optimization Algorithm

The Mixed Bayesian Optimization Algorithm (MBOA) [11] uses a set of decision trees to express the local structures of Bayesian network. Its implementation was described in detail in [11]. It is suitable for both continuous and/or discrete optimization problems. In this paper we propose its parallel version. The target architecture for parallel MBOA is the cluster of workstations connected with the hardware switch.

4.1 Parallel decision trees construction

Some principles identified in section 3 are useful for MBOA too. Namely in the model learning step the concept of restricted ordering of nodes from DBOA can be used again to keep the dependencies acyclic and to remove the need for communication during parallel construction of the probabilistic model. An ordering permutation $(o_0, o_1, ..., o_{n-1})$ means that only the variables $\{X_{o_0}, ..., X_{o_{i-1}}\}$ can serve as splits in the binary decision tree of target variable X_{o_i}. The drawback of predetermined ordering of nodes is that each tree can be constructed independently of the other trees, which allows for efficient parallelization. Moreover, it significantly reduces memory requirements for temporary data structures because only one decision tree is constructed at each time. In binary case the simplified code for building one decision tree is:

```
function BuildTree(Population Pop, TreeIndex i,
                              Permutation o): DecisionTreeNode;
begin
    for j:=0 to i-1 do
      if X[o[j]] has not been used as the split in i-th tree yet
        Evaluate the gain of X[o[j]] split for X[o[i]] target;
    Pick the split X[o[j'] with the highest metrics gain;
    if model complexity penalty  >  gain of X[o[j']
        return new UnivariateProbabilityLeaf(X[o[i]],Pop);
    Pop1 := SelectIndividuals (Pop,"X[o[j']] = 0");
    Pop2 := SelectIndividuals (Pop,"X[o[j']] = 1");
    return new SplitNode(new SplitVariable(X[o[j']]),
                 BuildTree(Pop1,i,o), BuildTree(Pop2,i,o));
end
```

4.2 Parallel decision trees sampling

In MBOA the decision trees are used also for continuous or mixed domains, so the decision nodes in the trees can have parameters of various types: real numbers for splitting on real-coded parameters, sequences of integers for splitting on categorical parameters, etc. Also the leaf nodes in the trees can contain various structured parameters: a list of Gaussian kernels, a list of Gaussian network coefficients, probability bias for binary leaves, etc.

It is a very time consuming task for each process to communicate the whole decision tree. It would be far more natural if each decision tree could be used exactly at the same workstation where it has been built. Thus, the goal of our design is to propose the parallel MBOA that uses the "horizontal" layout for offspring generation (see Fig. 1 left), but we would like parallel MBOA to be suited for coarse-grained platform, where the long communication latencies force us to use the "vertical" layout. To solve this problem, we interlace the model building task with the offspring generation task.

Each workstation uses two threads—a high priority thread for building the decision tree and a low priority thread for sampling previously built decision trees. The model building thread has to communicate with the master workstation to get the job and the model sampling threads have to communicate with other workstations to exchange the generated parts of offspring. By switching the threads it is possible to avoid most of the communication delays.

4.3 Multithreaded processing & TRANSIM tool

To switch the threads automatically and effectively, we presume workstations with multithreading capabilities. In simultaneous multithreading (SMT) one or more additional threads can take up the slack when the processor cannot continue execution for any reason. Since this is done by switching the active register set, without the need to copy the registers off to slower memory, the switch can be very fast. SMT capabilities, while not giving the performance of multiple processors, are very inexpensive to incorporate into the processor, because the execution units need not be replicated.

Although these days SMT becomes very popular—especially with the new Intel's Hyper-Threading(HT) technology—the current implementation of MPI standard does not support multithreaded processing to implement Multithreaded MBOA. Another possibility was to use the multitasking nature of Unix system, but we were looking for a concept at higher level of abstraction.

Finally, we decided to simulate the multithreaded MBOA using the well known TRANSIM tool. TRANSIM is a CASE tool used in the design of parallel algorithms. Its major function is the prediction of time performance early in the life cycle before coding has commenced. The TRANSIM language is a subset of transputers' Occam language with various extensions. It is not intended as a general purpose programming language, but to provide a control structure whereby the performance properties of an application may be expressed. Parallel execution, alternation, channel communication, time-slicing, priorities, interruption, concurrent operation of links and the effects of external memory are taken into account.

4.4 Multithreaded MBOA design

The whole architecture of multithreaded MBOA is shown in Fig. 2. The *farmer* thread is responsible for dynamic workload assignment, *builder* threads are responsible for building decision trees and *generator* threads are responsible for generating new genes. *Buffer* threads are used only for buffering dataflow between builders and generators, because TRANSIM does not implicitly support buffered channels. Threads $builder_i$, $buffer_i$ and $generator_i$ are mapped to the same i-th workstation.

The processes in TRANSIM communicate through pairwise unidirectional channels. The array of external channels *ch.master.in* is used by *farmer* to receive the job requests and the array of external channels *ch.master.out* is used to send the jobs to *builders*. A two-dimensional array of external channels *ch.network* is used to transfer the partially generated population between *generators*. The internal channels *ch.buffer.in*, *ch.buffer.req* and *ch.buffer.out* are used for exchange of local parts of model (decision tree) between *builder* and *generator*.

The principles of the most important threads *builder* and *generator* will be illustrated by fragments of TRANSIM input file. For those readers non-familiar with the statements of TRANSIM input language the non-intuitive parts will be re-explained afterwards. The complete code can be downloaded from web page *http://jiri.ocenasek.com/*.

First see the code fragment of the *builder* thread:

```
PLACED PAR i=0 FOR NUMWORKERS
  INT prev,next,j,prefetch.prev,prefetch.next,prefetch.j:
  SEQ | builder
    ch.farmer.in[i]  ! i | REQ.LENGTH
    ch.farmer.out[i] ? prefetch.j
    ch.farmer.out[i] ? prefetch.prev
    ch.farmer.out[i] ? prefetch.next
    WHILE prefetch.j < n
      SEQ
        j    := prefetch.j
        next := prefetch.next
        gene := prefetch.gene
        PRI PAR
          SEQ | getjob
            ch.farmer.in[i]  ! i | REQ.LENGTH
            ch.farmer.out[i] ? prefetch.j
            ch.farmer.out[i] ? prefetch.prev
            ch.farmer.out[i] ? prefetch.next
          SEQ | build
            ch.buffer.in[i] ! prev | WORK.LENGTH
            SERV(j*TIMESLICE)
            ch.buffer.in[i] ! j | WORK.LENGTH
            ch.buffer.in[i] ! next | WORK.LENGTH
```

Each *builder* requests for a job via *ch.farmer.in[i]* and it receives job info from *ch.farmer.out[i]*. More precisely, it receives the job number j, the number of workstation working on job $j-1$ and the number of workstation requesting the farmer for job $j+1$.

The useful computation is simulated by a *SERV()* command. You see that for fixed population size the time for building the decision tree for gene X_{o_j} is proportional to j and is scaled by *TIMESLICE* constant, according to empirical observations on sequential MBOA. The building of the decision tree is overlapped by high priority communication (see the sequence named "*getjob*") which serves for prefetching the next job from *farmer*.

Now see the code of *generator* thread. This fragment is very simplified, ignoring the initial stage of generating independent genes and the final stage of broadcasting the full population:

```
PLACED PAR i=0 FOR NUMWORKERS
  INT from,to,population,j:
  SEQ | generator
    WHILE TRUE
      SEQ
        ch.buffer.req[i] ! i | WORK.LENGTH
        ch.buffer.out[i] ? from
        PAR
          SEQ | recvpred
            ch.network[from][i] ? population
          SEQ | recvmodel
            ch.buffer.out[i] ? j
            ch.buffer.out[i] ? to
        SERV(GENSLICE)
        ch.network[i][to] ! population | j * N
```

Each *generator* receives from *builder* (via buffered channels) the number of workstation from which it then receives the population with genes $\{X_{o_0}, ..., X_{o_{j-1}}\}$ generated. Simultaneously it also receives from *builder* the decision tree for gene X_{o_j} and the number of consecutive workstation. Then the *SERV()* command simulated the sampling of the decision tree. According to empirical observations the sampling time is almost constant for fixed population size, independent of j. At the end, the population with generated $\{X_{o_0}, ..., X_{o_j}\}$ is sent to a consecutive workstation. To lower the communication demands further, it is sufficient to communicate only genes $\{X_{o_{k+1}}, ..., X_{o_j}\}$ if it exists some X_{o_k} that was previously generated in that consecutive workstation.

Note that the priority of threads is crucial. The *generator* thread should have high priority and the workstation processor should use the *builder* threads to overlap the communication latency between generators.

5 Simulation results

The simulation parameters were set according to the parameters measured on the real computing environment - a cluster of Athlon-600 computers connected by the hardware multiport router Summit48 (from Extreme Networks). The external communication is supposed to be the MPI (Message Passing Interface) which provides the well known standard for message passing communication.

TRANSIM is able to simulate the capacity of real communication channels, so the speed of external channels was set to 100Mb/s (like the speed of Summit48 Ethernet switch) and their communication latency was set to 0.5 ms (like the average software latency in MPI). The speed of internal channels was set to

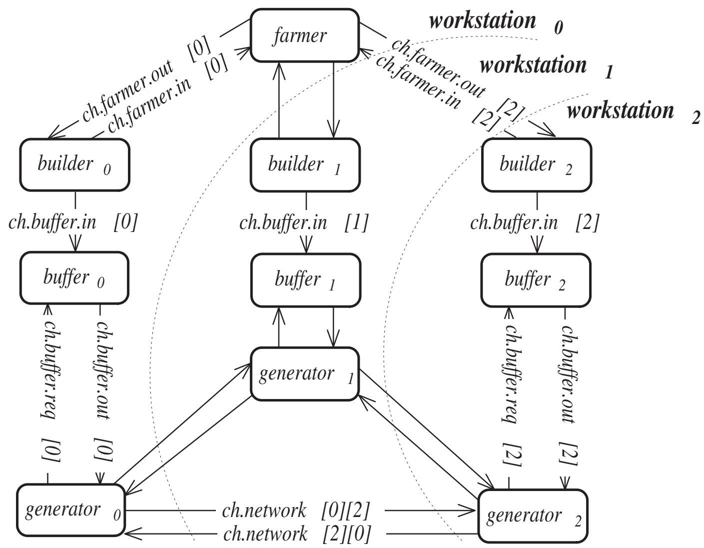

Fig. 2. Architecture of multithreaded MBOA in TRANSIM. An example with 3 workstations. The dashed curves separate processes mapped to different hardware.

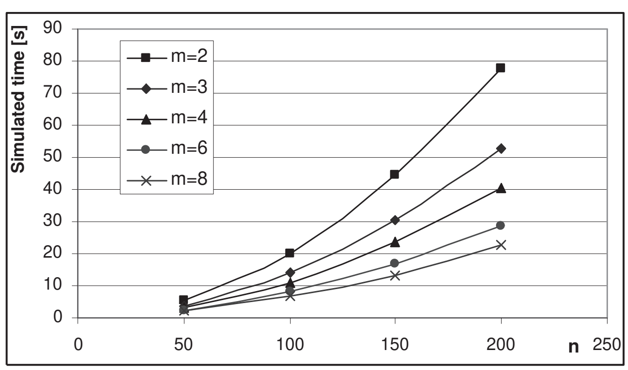

Fig. 3. The total simulated time of Multithreaded MBOA for varying problem size n and varying number of processors m for fixed population size $L = 2200$.

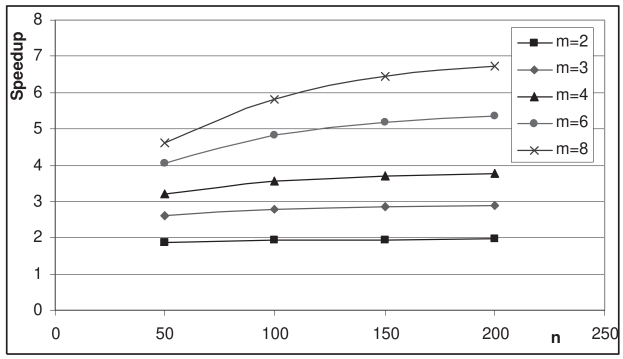

Fig. 4. The speedup of simulated multithreaded MBOA for varying problem size n and varying number of processors m for fixed population size $L = 2200$.

maximum, because in the real implementation the *builder* and *generator* would communicate via shared memory space. According to our empirical experience with sequential MBOA on Athlon-600 processor, the TIMESLICE constant was set for about 7.5 ms and the GENSLICE was set for 10 ms. The job assignment from *farmer* thread does not need extra communication capacity, so the message-length constants WORK.LENGTH and REQ.LENGTH were set to 1.

Fig. 3 depicts the execution time of parallel MBOA with respect to the problem size and number of processors. In Fig. 4 the results indicate that the achieved speedup slightly differs from ideal case because it is impossible to remove all communication latencies completely (for example there remain no decision trees to build when generating the last gene $X_{o_{n-1}}$). Fortunately, this difference becomes negligible with increasing problem size, so our approach is linearly scalable.

6 Summary and conclusions

We proposed and simulated the Multithreaded Mixed Bayesian Optimization Algorithm (MMBOA). We focused mainly on the effective construction of Bayesian networks with decision trees in the distributed environment. MMBOA is especially suitable for running on a cluster of workstations. Where available, additional efficiency can be achieved by utilizing multithreaded processors.

The results of simulation indicate that our algorithm is efficient for a large number of processors and its scalability improves with an increasing problem size. Moreover, the workload balance of each distributed process is adapted according to its computational resources when different types of workstations are used.

The use of the proposed parallelization of decision tree construction is not limited to BOA or other EDAs; it can be used in machine learning, data mining, and other areas of artificial intelligence. The use of the proposed technique outside the scope of EC is a topic for future research.

Acknowledgement

The authors would like to thank the members of the Computational Laboratory ETH Zürich for valuable comments and discussions. This research had been partially supported by the Grant Agency of Czech Republic from research grant GA 102/02/0503 "Parallel system performance prediction and tuning".

References

1. Larrañaga, P.: A Review on Estimation of Distribution Algorithms. Estimation of Distribution Algorithms. A new Tool for Evolutionary Computation. P. Larrañaga, J. A. Lozano (eds.). Kluwer Academic Publishers, pp. 57-100, 2001.
2. Pelikan, M., Goldberg, D. E., Lobo, F.: A survey of optimization by building and using probabilistic models, IlliGAL Report No. 99018, University of Illinois at Urbana-Champaign, Illinois Genetic Algorithms Laboratory, Urbana, Illinois, 1999.
3. Bosman, P. A. N., Thierens, D.: An algorithmic framework for density estimation based evolutionary algorithms. Utrecht University Technical Report UU-CS-1999-46, Utrecht, 1999.
4. Ocenasek, J., Schwarz, J.: The Parallel Bayesian Optimization Algorithm, In: Proceedings of the European Symposium on Computational Inteligence, Physica-Verlag, Kosice, Slovak Republic, pp. 61-67, 2000.
5. Ocenasek, J., Schwarz, J.: The Distributed Bayesian Optimization Algorithm for combinatorial optimization, EUROGEN 2001 - Evolutionary Methods for Design, Optimisation and Control, Athens, Greece, CIMNE, pp. 115-120, 2001.
6. Lozano, J. A., Sagarna, R., Larrañaga, P.: Parallel Estimation of Distribution Algorithms. Estimation of Distribution Algorithms. A new Tool for Evolutionary Computation. P. Larrañaga, J. A. Lozano (eds.). Kluwer Academic Publishers, pp. 129-145, 2001.
7. Pelikan, M., Goldberg, D. E., Cantú-Paz, E.: BOA: The Bayesian optimization algorithm. Proceedings of the Genetic and Evolutionary Computation Conference GECCO-99, vol. I, Orlando, FL, Morgan Kaufmann Publishers, pp. 525-532, 1999.
8. Etxeberria, R., Larrañaga, P.: Global optimization using Bayesian networks. Second Symposium on Artificial Intelligence (CIMAF-99), Habana, Cuba, pp 332-339, 1999.
9. Mühlenbein, H., Mahnig, T.: FDA a scalable evolutionary algorithm for the optimization of additively decomposed functions. Evolutionary Computation, 7(4), pp. 353-376, 1999.
10. Pelikan, M., Goldberg, D. E., Sastry, K.: Bayesian Optimization Algorithm, Decision Graphs, and Occam's Razor, IlliGAL Report No. 2000020, University of Illinois at Urbana-Champaign, Illinois Genetic Algorithms Laboratory, Urbana, IL, 2000.
11. Ocenasek, J., Schwarz, J.: Estimation of Distribution Algorithm for mixed continuous discrete optimization problems, In: 2nd Euro-International Symposium on Computational Intelligence, Kosice, Slovakia, IOS Press, pp. 227-232, 2002.
12. Laumanns, M., Ocenasek, J.: Bayesian Optimization Algorithms for multi–objective optimization, In: Parallel Problem Solving from Nature - PPSN VII, Springer-Verlag, pp. 298-307, 2002.
13. Arst. R., Minsker, B. S., Goldberg, D. E.: Comparing Advanced Genetic Algorithms and Simple Genetic Algorithms for Groundwater Management. 2002 Water Resources Planning & Management Conference, Roanoke, VA, 2002.
14. Cantú-Paz, E.: Efficient and Accurate Parallel Genetic Algorithms. Boston, MA: Kluwer Academic Publishers. 2000.