THE DISTRIBUTED BAYESIAN OPTIMIZATION ALGORITHM FOR COMBINATORIAL OPTIMIZATION

Jiri Ocenasek
Brno University of Technology
Faculty of Electrical Engineering and Computer Science
Department of Computer Science and Engineering
Bozetechova 2, 61266 Brno, Czech Republic
Email: ocenasek@dcse.fee.vutbr.cz

Josef Schwarz
Brno University of Technology
Faculty of Electrical Engineering and Computer Science
Department of Computer Science and Engineering
Bozetechova 2, 61266 Brno, Czech Republic
Email: schwarz@dcse.fee.vutbr.cz

Abstract. The Bayesian Optimization Algorithm (BOA) belongs to the probabilistic model building genetic algorithms where crossover and mutation operators are replaced by probability estimation and sampling techniques. The learned Bayesian network BN as the most general probability model is used to encode the structure of solved combinatorial problems. In [1] we proposed and simulated the pipeline hardware architecture for BOA. The aim of this paper is to propose the distributed version of BOA algorithm with a coarse-grained parallelism. We focused primarily on the construction of Bayesian network in the distributed environment. In adittion, methods for overlapping the communication latency during generation, evaluation and broadcasting of new population among the processes are described. Much attention was devoted to the implementation of proposed approaches using a cluster of workstations as a computational platform.

Key words: genetic algorithm, estimation of distribution algorithm, Distributed Bayesian Optimization Algorithm, Bayesian network, dependency graph, cluster computing, coarse-grained parallelism.

1 INTRODUCTION

The Bayesian Network [2] is an oriented acyclic graph, its nodes correspond to discrete variables and edges represent dependencies between variables.
In such a way the graph encodes the overall probability as the product of multivariate conditional probabilities, example is shown in Figure 1.

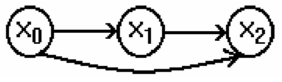

Figure 1 – The graphical representation of equation $p(X) = p(X_0)\,p(X_1|X_0)\,p(X_2|X_0X_1)$

The Bayesian Optimization Algorithm [3] operates on the population of strings/chromozomes of *n* binary variables/genes. Each generation it learns the Bayesian Network to encode the structure of promising solutions and then samples this network to effectively discovery the promising areas of the search space. The whole run of BOA can be described by the following pseudo-code:

```
Generate initial population of size $M$ (randomly);
Repeat
  Select parent population of $N$ individuals according to a selection method $(N \le M)$;
  Estimate the distribution of the selected parents by BN construction;
  Generate new offspring  of size $N'$ according to the estimated BN;
  Replace some individuals in current population by generated offspring;
Until termination criteria is met
```

The main goal of following chapters is to propose the BN construction and offspring generation using the Message Passing Interface standard (MPI).

2 PRINCIPLES OF SEQUENTIAL BOA

2.1 BN construction

To construct the Bayesian network a hill-climbing algorithm is used. It starts with an empty network and it subsequently adds the edge which maximizes the Bayes-Dirichlet metrics [2]. The BN construction is a very time consuming task with the complexity $O(n^3)$ - in sequential BOA it takes over 90% of the whole execution time. To keep the metrics computable, the dependency of each child gene (ending vertex) is limited to *k* parents (starting vertices).

2.2 BN sampling

After network construction the sufficient number of offspring for the new population is generated by the Probabilistic Logic Sampling (PLS). First, the

BN nodes are ordered in the topological order and each iteration, the nodes whose parents are already determined are generated using the conditional probabilities. This is repeated for each offspring until all its variables are generated.

3 DISTRIBUTED BOA

3.1 BN construction and composition

Because the BD metric is separable - can be written as a product of n factors, where i-th factor expresses the influence of edges ending in the variable X_i - we can propose the splitted network construction. Each parallel process is responsible for different subset of child nodes and the gain of each edge is computed as the local increase of BD metric (each process has its own local copy of whole parent population). It is possible to use up to n processes, each process corresponds to one child variable and it examines/adds only edges leading to this variable. In this case the BN construction time decreases roughly from $O(n^3)$ to $O(n^2)$.

The addition of edges is parallel, so we need an additional mechanism to keep the network acyclic. We proposed the concept of restricted set of parents. Each generation, nodes are ordered in advance, according to a random permutation vector $\boldsymbol{p}$. Each child node i may depend only on such parental nodes j having $p_j < p_i$. This approach ensures linear scalability, because no communication overhead is required to keep the network acyclic. The parallel greedy addition of edges can be described by the following pseudo-code:

```
generate random permutation vector $\boldsymbol{p}=(p_0,\ldots,p_{n-1})$ to restrict the parenthood
assign nodes to each of $m\leq n$ processes dynamically
**for** each node $i$ **do in parallel**
  compute the gain of edges from the set of restricted parents of node $i$
  **while** node_degree $d(i)\leq k$ **do**
    add the edge $(j\rightarrow i)$ with the highest score to the network B
    remove $j$ from set of restricted parents of node $i$
    recompute the gain of edges from remaining restricted parents of node $i$
  **end_while**
**end_for**
```

If $m<n$, we need to specify how to split the set of nodes among processes. Unfortunately, the number of restricted parents varies from node to node. To

keep all processors equally loaded, dynamic workload assignment is necessary. We implemented the farmer-workers architecture using a queue of child nodes to be processed. Process 0 acts as a farmer and also as a worker in the free time.
After BN construction each process contains different part of BN. To compose the whole network, we used the collective communication MPI_Allgather.

3.2 BN sampling and new population composition

We use the same PLS algorithm as for the sequential approach, but we arrange a defined number of processes, each of them generates a part of the new population. The amount of work is proportional to the number of new individuals in the sub-population to be generated, so we use static workload assignment. This assignment needs not be necessarily uniform, especially when non-homogenous cluster of workstations is used. We implemented the timing routines to measure the computational resources of each process and the results from the actual generation are used to modify the workload balance in the next generation.

The local generation of offspring is not time-excessive, but it is difficult to complete the huge population from the processes quickly. The standard blocking MPI_Allgather() routine can not be used because of performance degradation. We proposed and implemented a routine based on the pairwise communication:

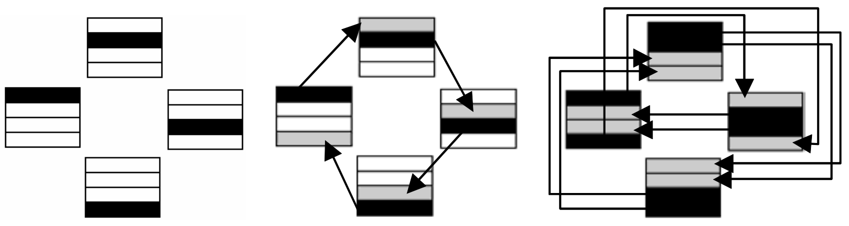

Figure 2 – The composition of the new population. The subpopulations generated or received in the previous step are marked black, the transferred subpopulations are gray. The communication distance and the number of transferred subpopulations in each step doubles, so the total number of steps is only log(P)!

This new routine is not atomic (it consists of fast non-blocking operations), so the subpopulation transfer can be overlapped by fitness computation. After that, the computed fitness sub-vectors can be broadcasted by the standard MPI_Allgather()(data amount is relatively small).

4 IMPLEMENTATION AND RESULTS

4.1 Workstation cluster topology

All experiments were run on the uniform cluster of Sun Ultra 5 workstations with UltraSPARC II processors at 270 MHz. The hardware multiport router has been Summit48 (from Extreme Networks), which is able to transfer up to 100 Mb/s between pairs of distinct workstations simultaneously. We used the MPI C++ library version 1.2.

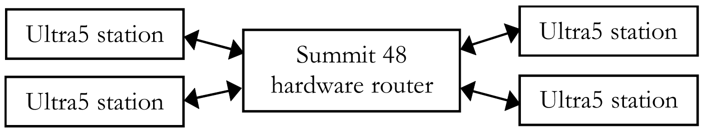

Figure 3 – The topology of the workstation cluster

4.2 Experimental results

For testing we used several deceptive functions and two combinatorial problems - knapsack problem and graph partitioning. We examined the performance of whole Distributed BOA and the performance of BN construction alone.

	P=1	P=2	P=3	P=4	P=5	P=6
Avg. execution time for 1 generation [s]	42.64	22.03	15.59	12.25	10.47	9.22
Speedup S	**1.00**	**1.94**	**2.73**	**3.48**	**4.07**	**4.63**
Efficiency 100S/P [%]	100	96.8	91.2	87.0	81.4	77.1
Avg. BN construction time [s]	40.97	20.91	14.19	10.88	9.00	7.81
Speedup S	**1.00**	**1.96**	**2.89**	**3.76**	**4.55**	**5.25**
Efficiency 100S/P [%]	100	98.0	96.3	94.1	91.0	87.4

Table 1 - The results for $f_{3\text{-deceptive}}$ function [3], chromozome length n=198, population size N=5000

We also found that for the case of the random graph partitioning the efficiency of the parallelization increases when the problem size grows (Fig. 4a) and that this efficiency is nearly not affected by the population size (Fig. 4b).

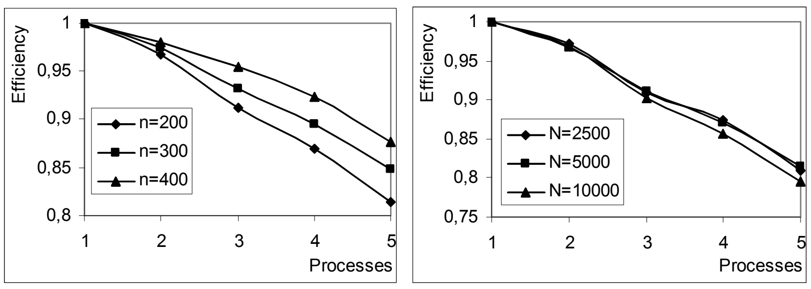

Figure 4a -Efficiency of the parallelization with the problem size n as a parameter (for fixed N=5000).
Figure 4b -Efficiency of the parallelization with the population size N as a parameter (for fixed n=200).

5 CONCLUSION AND FUTURE WORK

We proposed, implemented and successfully tested the distributed Bayesian Optimization Algorithm. We focused primarily on the effective construction of Bayesian network in the distributed environment. In addition, methods for overlapping the communication latency during generation, evaluation and broadcasting of the new population among the processes are described. Our algorithm is efficiently scalable, the computational time decreases almost linearly with the number of utilized workstations. Moreover, the workload balance of each distributed process is adapted according to its computational resources when different types of workstations are used in the cluster.

The proposed approaches can be also used out of the evolutionary optimization - some of them can be adapted for data-mining tasks. The next work will be directed to the multiobjective optimization algorithm [4] and its modification [5] to get the Distributed Multiobjective BOA.

Acknowledgement: This research has been carried out under the financial support of the Czech Ministry of Education – FRVS No. 0171/2001 "The parallelization of Bayesian Optimization Algorithm" and the Research intention No. CEZ: J22/98: 262200012 – "Research in information and control systems".

6 REFERENCES

[1] Ocenasek, J., Schwarz, J.: *The Parallel Bayesian Optimization Algorithm*, Proceedings of the European Symposium on Computational Intelligence, Physica-Verlag, Kosice, Slovak Republic, 2000, pp. 61-67.

[2] Heckerman, D., Geiger, D., Chickering, M.: *Learning Bayesian networks: The combination of knowledge and statistical data.* Technical Report MSR-TR-94-09, Microsoft Research, Redmond, WA, 1994.
[3] Pelikan, M., Goldberg, D. E., Cantú-Paz, E.: *Linkage problem, distribution estimation, and Bayesian networks.* IlliGAL Report No. 98013, University of Illinois at Urbana-Champaign, Illinois Genetic Algorithms Laboratory, Urbana, Illinois, 1998.
[4] Zitzler, E.: *Evolutionary Algorithms for Multiobjective Optimization: Methods and Applications*, PhD thesis, Swiss Federal Institute of Technology (ETH), Zurich, Switzerland, November 1999.
[5] Schwarz, J., Ocenasek, J.: *Pareto Bayesian Optimization Algorithm for the Multiobjective 0/1 Knapsack Problem*, 7th International Mendel Conference on Soft Computing, Brno University of Technology, Faculty of Mechanical Engineering, Brno, 2001, pp. 131-136.