SINGLE-WALK PARALLELIZATION OF THE GENETIC ALGORITHM

Wojciech Bożejko
Wrocław University of Technology, Janiszewskiego 11-17, 50-372 Wrocław, Poland
wojciech.bozejko@pwr.wroc.pl

Mieczysław Wodecki
University of Wrocław, Joliot-Curie 15, 50-383 Wrocław, Poland
mwd@ii.uni.wroc.pl

Keywords: Metaheuristics, Parallel computing, Evolutionary algorithm.

Abstract: This paper aims at presenting theoretical properties which can be used to approximate the theoretical speedup of parallel genetic algorithms. The most frequently parallelization method employed to genetic algorithm implements a master-slave model by distributing the most computationally exhausting elements of the algorithm (usually evaluation of the fitness function, i.e. cost function calculation) among a number of processors (slaves). This master-slave parallelization is regarded as easy in programming, which makes it popular with practitioners. Additionally, if the master processor keeps the population (and slave processors are used only as computational units for individuals fitness function evaluation), it explores the solution space in exactly the same manner as the sequential genetic algorithm. In this case we can say that we analyze the single-walk parallel genetic algorithm.

1 INTRODUCTION

Metaheuristics based on the local search method can be presented as processes of a graph searching in which vertexes constitute points of the solutions space and arcs correspond to the neighborhood relation – they connect vertexes which are neighbors in the solutions space. We will call it *neighborhood graph*. For all NP-hard problems the related neighborhood has an exponential size. Moving on such a graph defines some *path* (or other: trajectory) in the solutions space. Parallel metaheuristic algorithms make use of many processes to parallel generating or searching the neighborhood graph.

One can define two approaches to parallelization of the local search process with the relationship to the number of trajectories which are concurrently generated in the neighborhood graph:

1. *single-walk parallelization* (single trajectory): fine-grained algorithms because of the need of fast communication (the most computational expensive parts of the algorithm are parallelized),
2. *multiple-walk parallelization* (many trajectories): coarse-grained algorithms, communication takes place more rarely, comparing to single-walk parallelized algorithms.

These approaches challenge from the algorithm some requirements of communication and synchronization frequency, which implies the kind of granularity. Single-walk parallel metaheuristics are usually fine-grained algorithms (Bożejko et al., 2010; Bożejko et al., 2008b), multiple-walk metaheuristics – coarse-grained (Bożejko et al., 2008a).

1.1 Single-walk Parallel Algorithms

Single walk algorithms go along the single trajectory, but they can use multithread calculations to the neighborhood decomposition or parallel cost function computation. For example, calculations of the cost function value for more complicated cases are frequently equivalent in determining the longest (critical) path in a graph, as well as maximal or minimal flow.

1.2 Multiple-walk Parallel Algorithms

Algorithms which make use of a multithread multiple-walk model search concurrently a solution space by parallel working searching threads. Additionally, these algorithms can be divided into sub-

classes due to communication among threads (information about actual searching state):

1. independent search processes,
2. cooperative search processes.

If the multithread application (i.e. concurrently working search processes) does not exchange any information we can talk about *independent* processes of search. However, if information accumulated during an exploration of the trajectory is sent to another searching process and used by it, then we can talk about *cooperative* processes (Bożejko et al., 2008a). We can also come across a mixed model, so-called *semi-independent* (Czech, 2002) executing independent search processes keeping a number of common data.

2 THE METHODOLOGY OF METAHEURISTICS PARALLELIZATION

The majority of practical artificial intelligence issues, especially connected with planning and jobs scheduling, belongs to the class of strongly NP-hard problems, which require complex and time-consuming solution algorithms. Two main approaches are used to solve these problems: exact methods and metaheuristics. From one side, existing exact algorithms solving NP-hard problems possess an exponential computational complexity – in practice they are extremely time-consuming. From the other side, metaheuristics provide with suboptimal solutions in a reasonable time, even being applied in real-time systems.

Quality of the best solutions determined by approximate algorithms depends, in most cases, on the number of analyzed solutions, therefore on the time of computations. Time and quality demonstrates opposite tendency in the sense that to obtain a better solution requires significant increase of computations time. Parallel algorithms construction makes it possible to increase significantly the number of considered solutions (in a unit of time) effectively using multiprocessor computing environment.

The process of an optimization algorithm parallelization is strongly connected with the solution space search method used by this algorithm. The most frequent are the two following approaches: *exploitation* (or *search intensification*) and *exploration* (or *search diversification*) of the solution space. Due to this classification we can consider major categories of the metaheuristic class such as: local search methods (i.e. tabu search TS, simulated annealing SA, greedy randomized adaptive search procedure GRASP, variable neighborhood search VNS) and population-based algorithms (i.e. genetic algorithm GA, evolutionary strategies ESs, genetic programming GP, scatter search SS, ant colony optimization ACO, memetic algorithm MA, estimated distribution algorithms EDAs). Local search methods (LSM) start with a single initial solution improving it in each step by neighborhood searching. LSMs often find a locally optimal solution – they are focused on the solution space *exploitation*. Population-based algorithms (PBAs) use a population of individuals (solutions), which is improved in each generation. It means that the average goal function of the whole population usually improves itself – it does not equal improving of all the individuals. The whole process is randomized, so these methods are almost always nondeterministic. We can say that PBAs are focused on the solution space *exploration*.

3 POPULATION-BASED ALGORITHMS

Population-based algorithms (genetic, memetic, particle swarm optimization, etc.) are well-suited to parallelization due to its natural partitioning onto separate groups of solutions, which are concurrently processed. The method of using population of individuals allows us to *diversify* searching process onto the whole solution space. On the other hand, using cooperation, it is easy to *intensify* the search after finding a good region by focusing individuals onto it. Thanks to its concurrent nature, population-based algorithms are very handy to parallelize, especially in the independent way using multi-start model. Low level parallelization is not so easy because special properties of the considered problem have to be usually used.

3.1 Genetic Algorithm

Genetic Algorithm (GA) method is an iterative technique that applies stochastic operators on a set of individuals (population). Each individual of the population encodes the complete solution. Starting population is usually generated randomly. A GA applies a recombination operator (crossover) on two solutions in order to introduce diversity of population. Additionally, a mutation operator which randomly modifies an individual is applied as the insurance against stagnation of the search process. Traditionally GAs were associated with the binary representation of a solution, however in jobs scheduling area a permutational solution representation is more popular and useful.

The performance of population-based algorithms, such as GAs, is specially improved when running concurrently. Two strategies of parallelization are commonly used:

1. computations parallelization, in which operations allied to each individuals (i.e. goal function or genetic operators) are performed in parallel, as well as
2. population parallelization in which the population is partitioned into different parts which can be evolved concurrently or exchanged.

We distinguish the following kinds of parallelization techniques which are usually applied to genetic algorithms:

- *Global parallelization.* This model is based on the master-slave type concurrent processes. The calculations of the objective function are distributed among several slave processors while the main loop of the genetic algorithm is executed by the master processor.
- *Independent runs.* This approach runs several versions of the same algorithm with different parameters on various processors, allowing the parallel method to be more efficient. The independent runs model can be also considered as the distribution model without migration.
- *Distributed (island) model.* This model assumes that a population is partitioned into smaller subpopulations (islands), for which sequential or parallel GAs (usually with different crossover and mutation parameters) are executed. The main characteristic of this model is that individuals within a particular island can occasionally migrate to another island.
- *Cellular (diffusion) model.* In this model the population is mapped onto neighborhood structure and individuals may only interact with their neighbors. The neighborhood topology is usually taken from the physical processors connection network, so this is a fine-grained parallelism where processors hold just a few individuals.

The distribution model is the most common parallelization of parallel GAs since it can be implemented in distributed-memory MIMD machines, such as clusters and grids. This approach follows to coarse-grain parallelization (Bożejko and Wodecki, 2006). Fine-grained parallel implementations of the cellular (also called diffusion) model are strongly associated to the machines on which they are executed (Davidor, 1991). Master-slave implementations are available, also as general frameworks (i.e. ParadisEO of Cahon et al. (Cahon et al., 2004)).

We present two approaches in this chapter. The first one, in the Section 4, follows from Cantú-Paz (Cantú-Paz, 2005) and we discuss it briefly. The second one, described in the Section 5, constitutes a new idea of the broadcasting time approximation for the master-slave parallel genetic algorithm.

4 SEQUENTIAL BROADCASTING

A parallel genetic algorithm based on the master-slave model consists of two major modules: (1) communication module, performed chiefly by the master processor which broadcasts a part of population among slave processors, and (2) computations modules, executed both on master and slaves, in which evaluation of the fitness function is performed. We use a notation, taken from Cantú-Paz (Cantú-Paz, 2005). Let T_c be a time used to send a portion of data between two processors, and let T_f denote the time required to evaluate one individual. Each of processors, i.e. both master and slaves, evaluates a fraction of the population in the time $\frac{nT_f}{p}$, where p is the number of processors and n is the size of the population. Next, we assume in this section that the master broadcasts the data to slaves processors sequentially, as Figure 1 shows. We omit the time consumed by genetic operators as well as by the mutation (it is usually much shorter than the time of the fitness function evaluation). We also assume that the part of data assigned to each processor (i.e. the number of individuals evaluated) is the same both for each slave processor, and for the master processor.

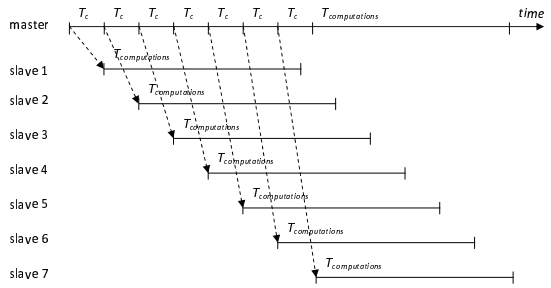

Figure 1: Sequential broadcasting in the master-slave parallel genetic algorithm.

For a sequential model of broadcasting, the parallel running time is given by the equation

$$T_p = pT_c + \frac{nT_f}{p}. \quad (1)$$

Let us check for which p the T_p is minimal. We de-

notes this p by p_1^*. Calculating $\frac{\partial T_p}{\partial p} = 0$ we get

$$T_c - \frac{nT_f}{p^2} = 0, \tag{2}$$

$$p = p_1^* = \sqrt{\frac{nT_f}{T_c}}, \tag{3}$$

which provides us with an optimal number of processors p_1^* minimizing the value of the parallel running time T_p. Calculating the maximum value of the theoretical speedup S_p we obtain

$$S_p = \frac{T_s}{T_p} = \frac{nT_f}{pT_c + \frac{nT_f}{p}}. \tag{4}$$

Substituting the optimal number of processors p_1^* we have

$$S_{p_1^*} = \frac{nT_f}{p_1^* T_c + \frac{nT_f}{p_1^*}} = \frac{nT_f}{\sqrt{\frac{nT_f}{T_c}}T_c + \frac{nT_f}{\sqrt{\frac{nT_f}{T_c}}}} =$$

$$= \frac{nT_f}{\sqrt{nT_f T_c} + \sqrt{\frac{(nT_f)^2}{\frac{nT_f}{T_c}}}} = \frac{\sqrt{(nT_f)^2}}{2\sqrt{nT_f T_c}} = \frac{1}{2}\sqrt{\frac{nT_f}{T_c}} = \frac{1}{2}p_1^*, \tag{5}$$

which gives us a maximal possible speedup for this model of the single-walk master-slave parallel genetic algorithm.

Figure 2 shows possible theoretical speedups for a given ratio $g = \frac{T_f}{T_c}$. The speedup is plotted for $g = 1, 2, 4$ showing that linearity of the speedup increases with g parameter. In practice, T_f is much greater than T_c. In such a situation the parallel algorithm can achieve near-linear speedup for the number of processors from the range $[1, p_1^*]$. For the number of processors greater than p_1^* speedup quickly decreases.

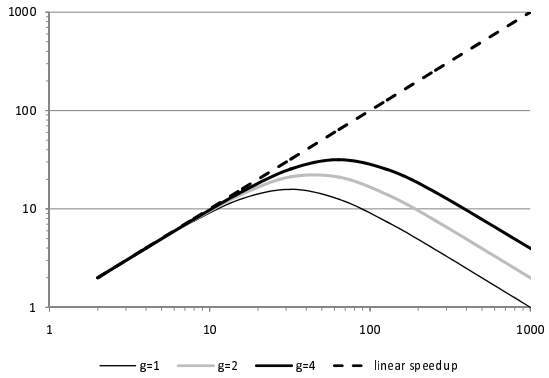

Figure 2: Theoretical speedups for the *sequential broadcasting* in the master-slave parallel genetic algorithm.

5 TREE-BASED BROADCASTING

Now, we propose faster model of communication for the master-slave parallel genetic algorithm. The broadcasting process is based on tree communication scheme, which gives us a possibility to obtain logarithmic complexity of the broadcasting process. This broadcasting scheme needs cooperation of all processors during the communication process. As scheme of the master-slave parallel genetic algorithm based on this communication model is shown on the Figure 3.

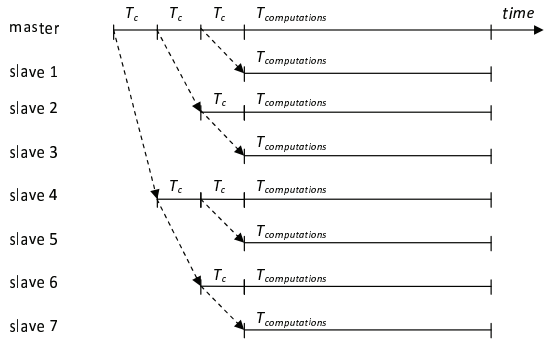

Figure 3: Tree-based broadcasting in the master-slave parallel genetic algorithm.

For the tree-based communication model the parallel running time T_p is estimated by

$$T_p = T_c \log p + \frac{nT_f}{p}. \tag{6}$$

In the case of using more processors, the parallel computations time $(\frac{nT_f}{p})$ decreases, whereas the time of communication $(T_c \log p)$ increases. We are looking for such a processors number p (let us call it p_2^*) for which T_p is minimal. Calculating $\frac{\partial T_p}{\partial p} = 0$ we obtain

$$\frac{T_c}{p} - \frac{nT_f}{p^2} = 0 \tag{7}$$

and then

$$p = p_2^* = \frac{nT_f}{T_c}, \tag{8}$$

which provides us with an optimal number of processors p_2^* which minimizes the value of the parallel running time T_p for this model of broadcasting. Calculating the maximum value of the theoretical speedup S_p we have

$$S_p = \frac{T_s}{T_p} = \frac{nT_f}{T_c \log p + \frac{nT_f}{p}}. \tag{9}$$

Substituting the optimal number of processors p_2^* we obtain

$$S_{p_2^*} = \frac{nT_f}{T_c \log p_2^* + \frac{nT_f}{p_2^*}} = \frac{nT_f}{\sqrt{T_c \log \frac{nT_f}{T_c}} + \frac{nT_f}{\frac{nT_f}{T_c}}} =$$

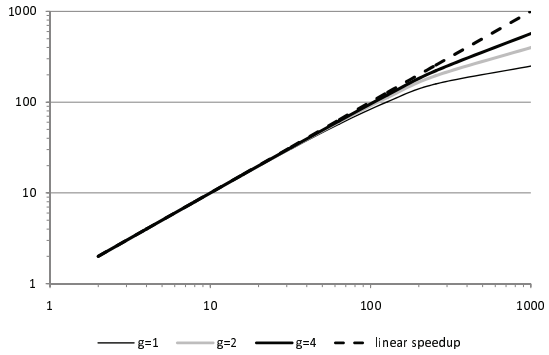

Figure 4: Theoretical speedups for the *tree-based broadcasting* in the master-slave parallel genetic algorithm.

$$= \frac{nT_f}{T_c(1+\log\frac{nT_f}{T_c})} = \frac{p_2^*}{1+\log p_2^*}. \qquad (10)$$

This equation provides us with a maximal possible speedup for the tree-base model of broadcasting for the single-walk master-slave parallel genetic algorithm.

The Figure 4 shows possible theoretical speedups for a given ratio $g = \frac{T_f}{T_c}$, $g = 1,2,4$. As for sequential communication plotted on the Figure 2, linearity of the speedup increases with the increase of the g parameter. The parallel algorithm achieves the near-linear speedup for the number of processors from the range $[1, p_2^*]$. For the number of processors greater than p_2^* speedup keeps on increasing.

6 REMARKS AND CONCLUSIONS

In this paper we discussed some theoretical properties of a metaheuristics which can be used to solve scheduling optimization problems. The tree-based broadcasting model seems to be more efficient than the sequential broadcasting model from the theoretical point of view. In practice, it is possible to make an additional improvement of the algorithm efficiency by fulfilling of some processors idle time during the communication phase – if the process is executed in the cycle, one generation of the parallel genetic algorithm after another, we can remove the synchronicity constraint. In such a case the master processor can execute a communication phase during a communication phase of the previous generation.

The proposed speedup estimation considered the parallel genetic algorithm based on the master-slave model of parallelism. The analyzed approaches give us a theoretical approximation of the optimal number of processors necessary to obtain the highest speedup. Additionally, it is possible to determine theoretical upper bounds for obtained speedups for the master-slave model of the parallel genetic algorithm with a single population kept by the master processor.

ACKNOWLEDGEMENTS

The work was partially supported by the Polish Ministry of Science and Higher Education, grants N N514 470439 (W. Bożejko) and N N514 232237 (M. Wodecki).

REFERENCES

Bożejko, W., Pempera, J., and Smutnicki, C. (2008a). Multi-thread parallel metaheuristics for the flow shop problem. In *Artificial Intelligence and Soft Computing (eds. L. Rutkowski, R. Tadeusiewicz, L.A. Zadeh, J.M. Zurada)*, pages 454–462. IEEE Computational Intelligence Society - Poland Chapter and the Polish Neural Network Society.

Bożejko, W., Pempera, J., and Smutnicki, C. (2008b). Parallel single-thread strategies in scheduling. In *Artificial Intelligence and Soft Computing*, volume 5097 of *Lecture Notes in Artificial Intelligence*, pages 995–1006. Springer.

Bożejko, W., Uchroński, M., and aw Wodecki, M. (2010). Parallel hybrid metaheuristics for the flexible job shop problem. *Computers & Industrial Engineering*, 59:323–333.

Bożejko, W. and Wodecki, M. (2006). A new inter-island genetic operator for optimization problems with block properties. volume 4029 of *Lecture Notes in Artificial Intelligence*, pages 324–333. Springer.

Cahon, S., Melab, N., and Talbi, E.-G. (2004). Paradiseo on condor-mw for optimization on computational grids. *http://www.lifl.fr/ cahon/cmw/index.html*.

Cantú-Paz, E. (2005). Theory of parallel genetic algorithms. In Alba, E., editor, *Parallel Metaheuristics*, pages 425–444. Wiley.

Czech, Z. (2002). Three parallel algorithms for simulated annealing. volume 2328 of *Lecture Notes in Artificial Intelligence*, pages 210–217. Springer.

Davidor, Y. (1991). A naturally occuring niche and species phenomenon: The model and first results. In *Proceedings of the Fourth Internatinal Conference of Genetic Algorithms*, pages 257–263.