RECENT TRENDS OF DEVELOPMENT IN EVOLUTIONARY OPTIMIZATION

Malay Kumar Pakhira
Member of Faculty, Department of Computer Science and Engineering
E-mail : malay_pakhira@yahoo.com

1. Introduction:

Evolutionary optimization techniques are considered to be very well known problem solving approaches nowadays. The term evolutionary optimization is coined from the famous Darwinian theory of natural evolution. After Mendelian rules of genetics are rediscovered by De Vries, T. Morgan and his associates completed the theory of evolutionary genetics. The processes of evolutionary optimization are reformulated by John Holland for possible applications in the fields of scientific computation that are related to problem solving via constrained optimization. Evolutionary optimization algorithms fall in the category of non-deterministic polynomial time algorithms and are most suitable for solving combinatorial optimization problems. In this article, we shall describe recent trends of developments in the field of evolutionary optimization.

2. Evolution Programs :

Although the era of evolutionary computation started with a humble beginning, very soon scientists realized its power and possibilities and started developing different possible manifestations of the same either by modification or by hybridization. In its simplest and most well known form we call it a genetic algorithm (GA). However, many other derivatives of the said algorithms have been developed, of which evolutionary programming (EP), evolution strategies (ES), genetic programming (GP) and different hybrids like annealing evolution are highly used in different applications. All these evolution techniques are categorized under the nickname of evolutionary algorithms (EA). Another stochastic combinatorial optimization technique developed and grown up parallely is simulated annealing (SA), which mimics the thermodynamic process of metallurgical cooling of molten solids. Although this procedure is not considered to be a member of the so called evolutionary algorithms, its components are borrowed for developing new evolutionary approaches. The above mentioned annealing evolution algorithm is one such hybrid derivative of an evolution program and simulated annealing.

2.1 Basics of SA and EA :

As mentioned earlier, both SA and EA are stochastic search and optimization techniques. The prime difference between an evolution program and simulated annealing is that while the former deals with a number of candidate solutions simultaneously, the later works with a single candidate solution in each step.

Now, we must describe what a candidate solution is. It is nothing but an assignment of values to the problem variables. Let, $f(x_1, x_2, \ldots, x_n)$ be the functional notation of the objective function of an optimization problem. Here, $x_1, x_2, \ldots, x_n$ are called problem variables. We need to find out the optimal assignment of values to these variables that will optimize the objective function. Before applying an evolutionary technique, it

should made confirmed that no deterministic polynomial time algorithm exists for solving the related problem.

In case of simulated annealing, one starts with a random assignment to the problem variables and then proceeds toward the optimal assignment by local searches. In each iteration a new candidate solution is generated by small local perturbation of variables which is accepted or rejected probabilistically depending on its figure of merit in optimizing the objective function. At the start of the process the perturbation mechanism makes long strides to explore the search space appropriately which is reduced gradually as the process cools down to a near optimal solution.

Now, since the search by simulated annealing in centered around a single candidate solution in each iteration, the process may produce only a locally optimal solution, i.e., may not reach the global solution. In other words, we can say that the search space explored by simulated annealing is very limited.

To overcome this local search problem, evolution programs start with a number of random candidate solutions, located at different positions in the search space.

Each of these candidate solutions then produce newer solutions by recombination and perturbation methods commonly known as crossover and mutation respectively. Figure of merit of each of the candidate solutions are then determined by evaluating the objective function, sometimes called metric, and better candidate solutions are selected for the next generation. The set of candidate solutions, used in a particular generation, is called a pool, and each candidate solution is termed a chromosome.

One common problem of both SA and EA is that both of these processes are very slow mainly due to the time-costly evaluation step. In technical terminology this problem is called the evaluation overhead or fitness callas. Both SA and EA maintain a predefined series of operations within a loop. In case of SA, these operations are candidate generation, temperature determination, fitness evaluation and candidate sampling. In case of EA, these are selection, crossover, mutation and evaluation. In both the cases, fitness evaluation is the most time consuming operation. Moreover, in case of EA, the evaluation overhead is compounded by the use of multiple candidate solutions.

3. Parallel SA and Parallel EA :

To speedup execution and to overcome the barrier of fitness evaluation, parallel versions of both SA and EA were implemented. In its parallel manifestation, SA almost looks like an EA. Different kinds of parallel versions of SA and EA are described in the following subsections.

3.1 Parallel Simulated Annealing:

Parallel SA procedures can be classified broadly on the basis of number of samplers used. In case a single sampler system, a particular candidate solution, out of a number of solutions produced by the generator, is sampled by the sampler for the next iteration. In case of a multiple sampler system, a number of samplers are used, each of which samples a particular solution and the sampled solutions are fed to the generators for production of the candidates for the next iteration. Schematic diagrams of the above systems are shown in Figures 1 and 2.

In both Figures 1 and 2, the generate units generate new candidate solutions by perturbation. The candidates are evaluated for their fitness at the metric evaluation units.

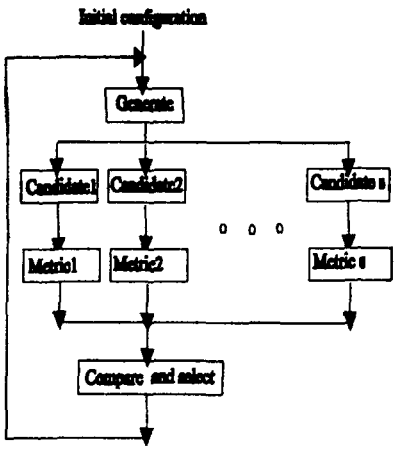

Figure 1: Parallel SA with single sampler

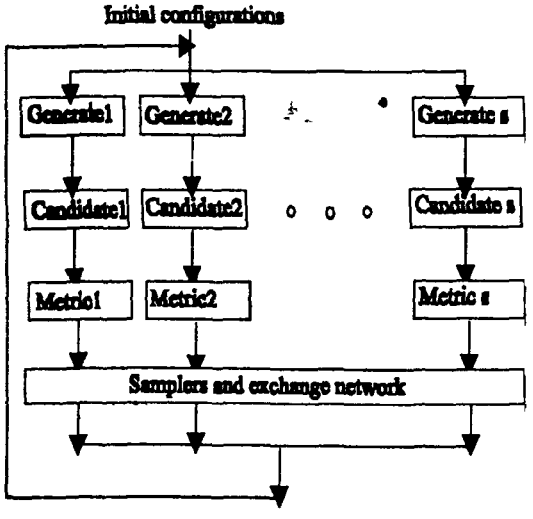

Figure 2: Parallel SA with multiple sampler

The sample exchange network then picks up suitable candidates for the next iteration. In case of a single sampler, only one of the candidates is picked up. And in case of multiple sampler a number of candidates are picked up and distributed among the generate units. The temperature scheduler part is not shown in the above figures. It is assumed a predetermined temperature schedule is maintained in the system.

We can implement a parallel SA either by simulation in a uniprocessor system or we can use a number of processors each of which processes only one candidate solution. Parallel versions are intended both for better solutions and better convergence rates. Now, since in SA, the number of iterations is very large, the metric evaluation units consume most of the processing time. Hence we can expect a really faster rate of execution only if we use a fast hardware evaluation unit. Another advantage of such an evaluation unit is that, we need not use a number of isolated processors for parallelization. A uniprocessor system connected to a number of hardware evaluation units will work fine. Also a pipeline of the processing stages is possible which enhance the speedup to a very high level.

3.2 Parallel Evolution Programs :

As in the case of SA, parallel versions of EAs also exist. Actually an EA is a natural candidate for parallelization. Evolution processes do have a lot of inherent parallelism within themselves. Exploration and exploitation of this inherent parallelism is only possible if we design parallel algorithms and the corresponding parallel hardware. Some of the parallel EA schemes are presented in Figures 3, 4 and 5. Parallel evolution programs differ mainly on the processor architecture and on the way the individual processors communicate among themselves. All of these architectures are supported by a distributed processing environment.

In case of the master slave processor architecture, the pool of chromosomes are sent to slaves, one to each processor, by the master. So the number of slave processors must be equal to the pool size. In the mesh or circular arrangements, the complete pool is divided into subpools of smaller sizes and distributed among

the processors, which execute EA processes of their own. After intervals of certain generations, evaluation units only instead of individual slave processors, we can make these parallel systems more cost-effective.

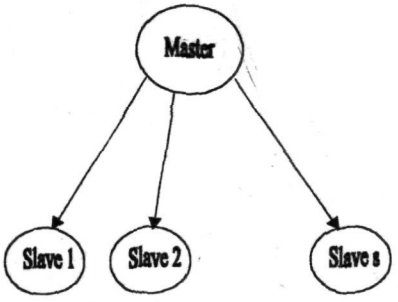

Figure 3: Master slave arrangement

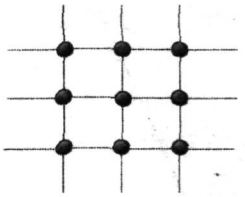

Figure 4: Mesh network of processors

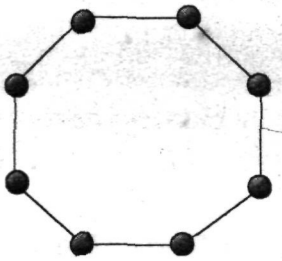

Figure 5: Cyclic arrangement of processors

information regarding intermediate candidate solutions are exchanged among the processors. In these cases also, each individual processor can employ their slaves for faster execution, which needs a large number of processors. Thus, if we can use specific general purpose hardware

3.3 Pipelined Evolution Programs:

Recently pipelined versions of different evolution programs are being developed. By use of pipelining we can get extra speedup by virtue of overlapped execution in time-space. A very simple pipeline using the operators of a conventional GA is shown in Figure 6.

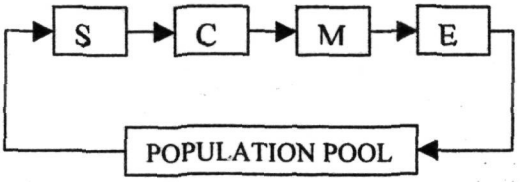

Figure 6: Basic pipeline architecture

Figure 7 shows a particular pipeline configuration for a sample problem. In this figure the number of processing elements used are

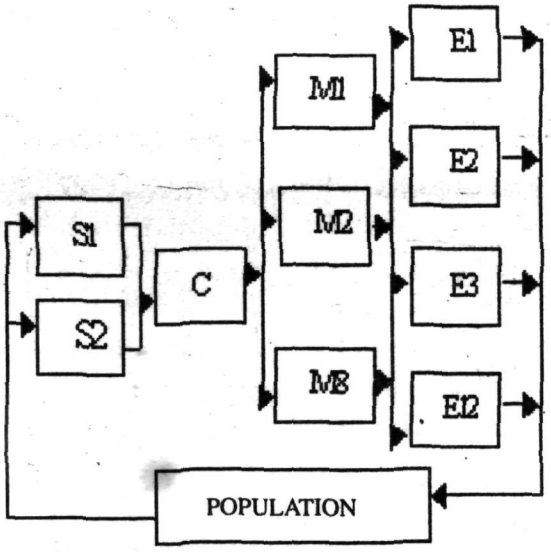

Figure 7: A sample pipeline configuration

determined by analyzing the relative complexities of the operations performed at these stages. In Figure 7, we have used 2 selection units (S), 1 crossover unit(C), 8 mutation units (M) and 12 evaluation units (E).

4. Conclusions:

An attempt is made, in this article, to give the reader an overview of recent trends of development in the fields of evolutionary optimization techniques. It is observed that one should try to develop reliable and faster parallel systems. Wherever possible, the bottlenecks must be removed by appropriate hardware components, and if possible, to develop a complete independent hardware platform. Replacement of costly multiprocessor by a uniprocessor and dedicated scalable hardware units is emphasized. Parallel hardware components may be suitable for pipeline processing. In the limited scope of this article one must not expect a detailed description of the underlying processes. However, interested readers may go through the references for better understanding.

References:

[1] J. Holland, "Adaptation in Neural and Artificial Systems", Ann. Arbor, MI: University of Michigan, 1975.

[2] D. E. Goldberg, "Genetic Algorithms in Search, Optimization and Machine Learning", New York, Addison-Wesley, 1989.

[3] Z. Michalewicz, "Genetic Algorithms + Data Structures = Evolution Programs", New York, Springer-Verlag, 1992.

[4] S. Kirkpatrik, C. D. Gelatt and M. P. Vecchi, "Optimization by Simulated Annealing", Science, vol. 220, pp. 671-680, 1983.

[5] E. Kantu-Paz, "A Survey of Parallel Genetic Algorithms", tech. Report, Illinois GA laboratory, Urbana, IL, 1997.

[6] M. K. Pakhira and R. K. De, "Function Optimization using a Pipelined Genetic Algorithm", in Proc. of Intl. Conf. on Intelligent Sensing, Sensor Networks and Information Processing (ISSNIP-04), Melbourne, Australia, pp. 253-257, 2004.

[7] M. K. Pakhira, R. K. De, S. Bandyopadhyay and U. Maulik, "Pipelined Processing of Genetic Clustering", in Proc. of Intl. Conf. on Intelligent Sensing and Information processing (ICISIP-04), Chennai, India, pp. 23-28, 2004 .

[8] S. Baluja, "Structure and performance of fine-grain parallelism in genetic search", in Proc. of the fifth Intl. Conf. on Genetic Algorithms, Morgan Kaufmann, San Mateo, CA, pp. 155-162, 1993.

[9] H. Muhlenbein, M. Scomisch and J. Born, "The Parallel Genetic Algorithm as Function Optimizer", in Proc. of the fourth Intl. Conf. on Genetic Algorithms, Morgan Kaufmann, San Mateo, CA, pp. 271-278, 1991.

[10] R. Azencott, "Simulated annealing : parallelization techniques", New York, John Wiley, 1992.

[11] S. Y. Lee and K. G. Lee, "Synchronous and asynchronous parallel simulated annealing with multiple Markov chains", IEEE Trans. On Parallel and Distributed Systems, vol. 7, no. 10, pp. 993-1008, 1996.

[12] M. K. Pakhira, "A Hybrid genetic algorithm using probabilistic selection", in journal of IE(I), vol. 84, pp. 23-30 2003.

[13] M. K. Pakhira, "The Power of Genetic Algorithms", in Reason (Technical Magazine of KGEC), pp 05 - 08, 2002.