PSSA: Parallel Stretched Simulated Annealing

Tiago Ribeiro[*], José Rufino[†] and Ana I. Pereira[**]

[*]*School of Technology and Management, Polytechnic Institute of Bragança, Portugal*
[†]*Department of Informatics and Communications, School of Technology and Management, Polytechnic Institute of Bragança, Portugal*
[**]*Department of Mathematics, School of Technology and Management, Polytechnic Institute of Bragança, Portugal*

Abstract. We consider the problem of finding all the global (and some local) minimizers of a given nonlinear optimization function (a class of problems also known as *multi-local programming problems*), using a novel approach based on Parallel Computing. The approach, named Parallel Stretched Simulated Annealing (PSSA), combines simulated annealing with stretching function technique, in a parallel execution environment. Our PSSA software allows to increase the resolution of the search domains (thus facilitating the discovery of new solutions) while keeping the search time bounded. The software was tested with a set of well known problems and some numerical results are presented.

Keywords: Simulated Annealing. Nonlinear Optimization. Global Optimization. Parallel Computing.
PACS: 02.60.Pn

INTRODUCTION

A multi-local programming problem aims to find all the global (and some local) solutions of the minimization problem

$$\min_{x \in X} f(x), \tag{1}$$

where $f : \mathbb{R}^n \to \mathbb{R}$ is a given multimodal objective function and X is a compact set defined by $X = \{x \in \mathbb{R}^n : a_i \leq x_i \leq b_i, i = 1, ..., n\}$.

So, the purpose is to find all global points $x^* \in X$ such that

$$\forall x \in X,\ f(x^*) \leq f(x), \tag{2}$$

and some local solutions, $\tilde{x}$, that satisfy

$$|f(x^*) - f(\tilde{x})| \leq \delta \tag{3}$$

for a positive value δ.

These problems appear in practical situations like ride comfort optimization [2], Chemical Engineering (process synthesis, design and control [3]), and reduction methods for solving semi-infinite programming problems [8, 14].

The most common methods for solving multi-local optimization problems are based on evolutionary algorithms, such as genetic [1] and particle swarm [10] algorithms. Additional contributions may be found in [7, 16, 18, 19].

Stretched Simulated Annealing (SSA) was also proposed [11, 12, 13] as a method to solve multi-local programming problems. SSA combines simulated annealing with stretching function technique in order identify the global minimizers and, if possible, some local solutions.

In this paper we introduce Parallel Stretched Simulated Annealing (PSSA) as a parallel version of the SSA method to compute all global (and some local) solutions of the minimization problem (1). PSSA was implemented in the C programming language [6], on top of an MPI (*Message Passing Interface*) [17] library. The implementation is able to explore the parallelism of modern multi-core systems, whether isolated or interconnected in a cluster configuration [15]. Basically, PSSA allows to increase the resolution of the search domains, while keeping the search time bounded.

The remaining of the paper is organized as follows. The section 2 describes the original SSA method and the new parallel approach to its application (PSSA), the Section 3 provides some numerical results produced by PSSA and the last section presents some conclusions and directions for future work.

PARALLEL STRETCHED SIMULATED ANNEALING

Sequential Stretched Simulated Annealing

The Stretched Simulated Annealing (SSA) method solves a sequence of global optimization problems in order to compute, in sequence, the local solutions of the minimization problem (1) that satisfy conditions (2) or (3). The objective function of each global optimization problem is obtained by applying a stretching function technique [9].

Let x_j^* be a particular solution. The mathematical formulation of the global optimization problem is as follows:

$$\min_{l_x \leq x \leq u_x} \Phi_l(x) \equiv \begin{cases} \hat{\phi}(x) & \text{if } x \in V_{\varepsilon^j}(x_j^*),\ j \in \{1,\ldots,N\} \\ \phi(x) & \text{otherwise} \end{cases} \tag{4}$$

where $\hat{\phi}(x)$ is defined as

$$\hat{\phi}(x) = \bar{\phi}(x) + \frac{\delta_2[\operatorname{sgn}(\phi(x) - \phi(x_j^*)) + 1]}{2\tanh(\kappa(\bar{\phi}(x) - \bar{\phi}(x_j^*))} \tag{5}$$

and

$$\bar{\phi}(x) = \phi(x) + \frac{\delta_1}{2}\|x - x_j^*\|[\operatorname{sgn}(\phi(x) - \phi(x_j^*)) + 1] \tag{6}$$

where δ_1, δ_2 and κ are positive constants, "sgn" is the well-known *sign* function and N is the number of minimizers already detected.

Parallel Approach

SSA searches for problem solutions in a certain given domain by following a stochastic strategy. In the original SSA implementation, this strategy is applied in l consecutive runs, where l is a parameter of the implementation. As l increases, the probability of finding new solutions also increases, as well as the computational time consumed.

Our approach to the parallelization of the SSA method is as follows: the search domain is subdivided in several subdomains (with the same amplitude) and the SSA method is individually applied to each subdomain; SSA can be applied to several subdomains at the same time, provided enough processors are assigned to work on the problem.

This parallelization strategy is not expected to translate into a smaller search time than the search in a single-domain and, in fact, it may considerable increase that time. The explanation is simple: although the subdomains are smaller than the original domain, each subdomain must still be stochastically processed in l consecutive runs (and although each subdomain is smaller, that may not be enough to counter-balance the need for l runs).

Thus, with PSSA we are primarily concerned in deepening the analysis of the original search domain by placing the same kind of computational effort into the analysis of each subdomain, and not to increase performance. However, with the right choice of the number of working processors (for a certain overall number of subdomains), we are at least able to ensure predictable (bounded) computational times, as we will show in the next section.

The parallel algorithm follows the well known master-slave pattern, with a static work division: a master process launches several slave processes; each slave is able to autonomously select an exclusive set of subdomains; these will be processed in order at each slave, and all slaves will work in parallel. We choose a static work division instead of a dynamic one because the number of subdomains may be potentially large (mainly for problems with a higher number of dimensions); this way, we prevent each slave of overloading the master with work requests (a subdomain would represent a work unit); the slaves communicate with the master only after having processed all of their subdomains; the master is responsible to process the byproducts of the slaves and to produce the final report of the computation.

This simple but highly efficient strategy was possible because of the "embarrassingly parallel" nature of the problem.

NUMERICAL RESULTS

For the computational tests we selected four problems presented in [4], known as Ackley, Branin, Levy and Hartmann. The coded algorithm terminates when it has not identified new solutions during some successive iterations. A limit on the number of iterations is also imposed. Other algorithm parameters are $\delta = 5.0$ and $l = 100$.

The numerical experiences were conducted in a small commodity cluster of 8 nodes, with a 2.6GHz quad-core processor (Intel Core2 Quad Q9400) per node. Each problem was studied in a subset of the following cluster configurations: a) 1 node (4 cores), b) 2 nodes (8 cores), c) 4 nodes (16 cores) and d) 8 nodes (32 cores). For each problem, and for each number of subdomains (N_s), the cluster configurations were chosen accordingly with the following criteria: i) try to ensure exactly one core per subdomain (that is, maximize parallelism), while minimizing the number of cluster nodes used; ii) if not enough cores are available to satisfy criteria i), use all the cluster cores. In any case, the number of MPI slave processes will always be equal to the number of cluster cores used.

The number of subdomains in each dimension follows an exponential (base 2) sequence that depends on the number of dimensions. So, for 2-dimensional problems (Ackley and Branin), we have $N_s = 2^{2\times i}$, for $i = 1, 2, 3$; accordingly, we have only considered the cluster configurations a), c) and d), respectively. For 3-dimensional problems (Hartman and Levy), we have $N_s = 2^{3\times i}$, for $i = 1, 2, 3$; thus, we have only considered the configurations b), d) and d), respectively.

Details of the numerical experiences are listed in the Table 1, where P refers to the problem name, N_s is the number of subdomains, N_c is the number of computing cores (or slave processes), N_n is the number of cluster nodes, $t(s)$ is the (average[1]) computing or search time (in seconds), f^* is the best solutions in all runs, x^* is the global minimizer and m is the number of other global, or local, solutions that satisfy the conditions (2) or (3). The results show that PSSA was able to identify a large number of global minimizers and some other local solutions.

TABLE 1. Numerical results

P	N_s	N_c	N_n	$t(s)$	f^*	x^*	m
Ackley	4	4	1	13.94	$8.412870\text{E}-11$	$(+9.747184\text{E}-12, -2.810112\text{E}-11)$	0
	16	16	4	14.73	$2.633098\text{E}-10$	$(-7.742246\text{E}-11, +5.169381\text{E}-11)$	0
	64	32	8	25.92	$5.333018\text{E}-10$	$(+1.189199\text{E}-11, +1.881754\text{E}-10)$	3
Branin	4	4	1	7.51	$3.978874\text{E}-01$	$(+9.424776\text{E}+00, +2.475001\text{E}+00)$	2
	16	16	4	8.89	$3.978874\text{E}-01$	$(+9.424778\text{E}+00, +2.474999\text{E}+00)$	2
	64	32	8	11.54	$3.978874\text{E}-01$	$(-3.141593\text{E}+00, +1.227500\text{E}+01)$	2
Hartmann	8	8	2	4.41	$-3.862782\text{E}+00$	$(+1.146695\text{E}-01, +5.556533\text{E}-01, +8.525475\text{E}-01)$	2
	64	32	8	8.27	$-3.862782\text{E}+00$	$(+1.144256\text{E}-01, +5.556620\text{E}-01, +8.525594\text{E}-01)$	4
	512	32	8	41.53	$-3.862782\text{E}-01$	$(+1.145690\text{E}-01, +5.556471\text{E}-01, +8.525513\text{E}-01)$	4
Levy	8	8	2	6.18	$9.267555\text{E}-10$	$(+1.000026\text{E}+00, +1.000012\text{E}+00, +9.999634\text{E}-01)$	1
	64	32	8	10.19	$2.066059\text{E}-10$	$(+9.999943\text{E}-01, +1.000005\text{E}-01, +1.000051\text{E}-01)$	5
	512	32	8	93.17	$6.101846\text{E}-11$	$(+1.000002\text{E}+00, +9.999913\text{E}-01, +9.999823\text{E}-01)$	8

We can verify that the search time of the PSSA implementation is bounded, as the number of subdomains increases. The stabilization of the search time is shown in Graphics 1.a) and 1.b).

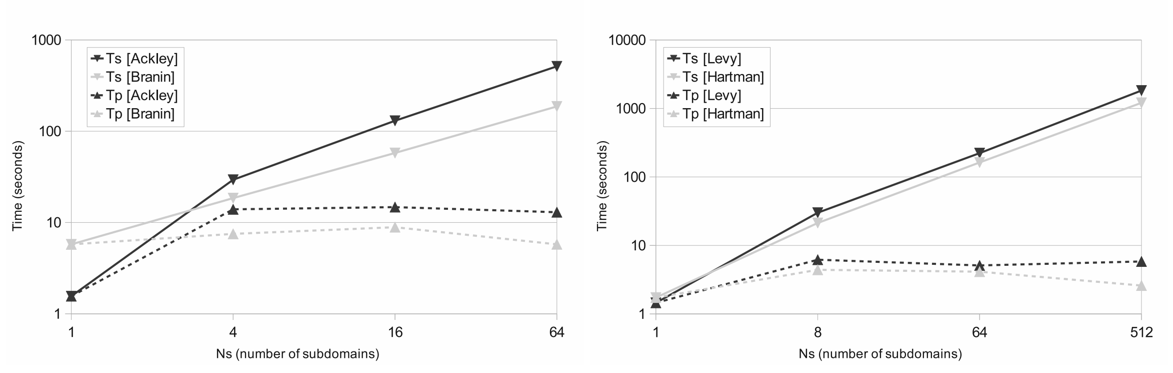

FIGURE 1. PSSA search times for a) 2-dimensional and b) 3-dimensional problems.

In the graphics, T_p is a search time derived from the t time of Table 1 as follows: for cluster configurations where each core is responsible for a single subdomain, $T_p = t$; for configurations where each core is responsible for $k = N_s/N_c$ subdomains (both Hartman and Levy, with 64 and 512 subdomains), we have $T_p = t/k$; in this case, T_p is not really an

[1] To accommodate possible variations of the computing time (due both to the stochastic nature of the algorithms or to transient instabilities of the cluster execution environment) the computing time is the arithmetic average of 10 executions of the same problem.

observed time, but a projected time based on the measured time t; the rationale is that if enough cores were available, each core would process one subdomain, and that would take approximately the same time per each subdomain.

The same graphics may also be used to derive a complementary conclusion: as subdomains get smaller, they tend to be processed more rapidly (despite each subdomain still taking l runs); this is the fundamental reason that explains the stabilization (and even a slight decreasing) of the overall search times in the PSSA method.

Finally, we also have studied the time T_s that would take to process all of the subdomains, in sequence, by running SSA (our previous approach) in a single core of our cluster. This study allowed us to perceive the real gains attained by PSSA when compared to SSA under the same circumstances. Thus, the previous graphics show that T_s grows exponentially with the number of subdomains, while T_p remains low and bounded as already observed.

CONCLUSIONS AND FUTURE WORK

In this work, we have introduced PSSA as a Parallel Computing based approach to the SSA stochastic algorithm, to find all global (and some local) solutions of multimodal objective function problems. The computational experiments showed that the PSSA algorithm is capable of locating all the global optima and a great number of some local solutions that satisfied condition (2) or (3). The experiments also demonstrated the scalability of our parallel approach.

In the future, we intend to use PSSA to deepen the study of the problems tackled in this paper and other representative problems of the literature.

REFERENCES

1. R. Chelouah and P. Siarry, A continuous genetic algorithm designed for the global optimization of multimodal functions, *Journal of Heuristics*, **6**, 191–213 (2000).
2. P. Eriksson and J. Arora, A comparison of global optimization algorithms applied to a ride comfort optimization problem, *Structural and Multidisciplinary Optimization*, **24**, 157–167 (2002).
3. C. Floudas, Recent advances in global optimization for process synthesis, design and control: enclosure of all solutions, *Computers and Chemical Engineering*, S963Ű-973 (1999).
4. A. R. Hedar,Global Optimization Test Problems, `http://www-optima.amp.i.kyoto-u.ac.jp/member/student/hedar/Hedar_files/TestGO.htm`
5. L. Ingber, Very fast simulated re-annealing, *Mathematical and Computer Modelling*, **12**, 967–973 (1989).
6. Kernighan and D. M. Ritchie, The C Programming Language (2nd ed.), Englewood Cliffs, NJ: Prentice Hall, ISBN 0-13-110362-8 (March 1988).
7. E. Kiseleva and T. Stepanchuk, On the efficiency of a global non-differentiable optimization algorithm based on the method of optimal set partitioning, *Journal of Global Optimization*, **25**, 209–235 (2003).
8. T. León, S. Sanmatías and H. Vercher, A multi-local optimization algorithm, *Top*, **Vol. 6**, N. 1, 1–18 (1998).
9. K. Parsopoulos, V. Plagianakos, G. Magoulas, and M. Vrahatis, Objective function stretching to alleviate convergence to local minima, *Nonlinear Analysis* **4**7, 3419–3424 (2001).
10. K. Parsopoulos and M. Vrahatis, Recent approaches to global optimization problems through particle swarm optimization, *Natural Computing*, **1**, 235–306 (2002).
11. A. I. Pereira, E. M. G. P. Fernandes, A reduction method for semi-infinite programming by means of a global stochastic approach, *Optimization*, **58**, 713–726 (2009).
12. A. I. Pereira, E. M. G. P. Fernandes, Constrained Multi-global Optimization using a Penalty Stretched Simulated Annealing Framework, *Numerical Analysis and Applied Mathematics, AIP Conference Proceedings*, **1168**, 1354–1357 (2009).
13. A. I. Pereira, E. M. G. P. Fernandes, Comparative Study of Penalty Simulated Annealing Methods for Multiglobal Programming, *2nd International Conference on Engineering Optimization* (2010).
14. C. Price, Non-linear semi-infinite programming, University of Canterbury, 1992.
15. T. Rauber and G. Runger, Parallel Programming for Multicore and Cluster Systems, Springer, ISBN 978-3-642-04817-3 (2010).
16. S. Salhi and N. Queen, A Hybrid Algorithm for Identifying Global and Local Minima When Optimizing Functions with Many Minima, *European Journal of Operations Research*, **155**, 51–67 (2004).
17. M. Snir, SW. Otto, S. Huss-Lederman and DW Walker, MPI-The Complete Reference (Volume 1), MIT Press, Cambridge, MA. ISBN 0-262-69215-5 (1988).
18. I. Tsoulos and I. Lagaris, Gradient-controlled, typical-distance clustering for global optimization, *www.optimization.org* (May 2004).
19. W. Tu and R. Mayne, Studies of multi-start clustering for global optimization, *International Journal Numerical Methods in Engineering*, **53**, 2239–2252 (2002).