Solving Constrained Multilocal Optimization Problems with Parallel Stretched Simulated Annealing

Ana I. Pereira[1,2](✉) and José Rufino[1,3]

[1] Polytechnic Institute of Bragança, 5301-857 Bragança, Portugal
{apereira,rufino}@ipb.pt
[2] Algoritmi R&D Centre, University of Minho, Campus de Gualtar, 4710-057 Braga, Portugal
[3] Laboratory of Instrumentation and Experimental Particle Physics, University of Minho, Campus de Gualtar, 4710-057 Braga, Portugal

Abstract. Constrained multilocal programming optimization problems may be solved by solving a sequence of unconstrained problems. In turn, those unconstrained problems may be solved using techniques like the Stretched Simulated Annealing (SSA) method. In order to increase the solving performance and make possible the discovery of new optima, parallel approaches to SSA have been devised, like Parallel Stretched Simulated Annealing (PSSA). Recently, Constrained PSSA (coPSSA) was also proposed, coupling the penalty method with PSSA, in order to solve constrained problems. In this work, coPSSA is explored to solve four test problems using the l_1 penalty function. The effect of the variation of the reduction factor parameter of the l_1 penalty function is also studied.

Keywords: Unconstrained optimization · Parallel computing

1 Introduction

Multilocal programming aims to identify all local maximizers of unconstrained or constrained nonlinear optimization problems. More formally, a constrained multilocal programming problem may be defined by the following formulation:

$$\begin{aligned} &\max f(x) \\ &\text{s.t. } h_k(x) = 0,\ k \in E \\ &\quad\ \ g_j(x) \leq 0,\ j \in I \\ &\quad\ \ -l_i \leq x_i \leq l_i,\ i = 1, \dots, n \end{aligned} \tag{1}$$

where at least one of the n-dimensional functions $f, h_k, g_j : \mathbb{R}^n \rightarrow \mathbb{R}$ is nonlinear, and E and I are index sets of equality and inequality constraints, respectively. Since concavity is not assumed, the nonlinear optimization problem can have many global and local (non-global) maxima. Consider the feasible region (search space) defined by $\mathcal{R} = \{x \in \mathbb{R}^n : -l_i \leq x_i \leq l_i, i = 1, \dots, n;\ h_k(x) = 0, k \in$

O. Gervasi et al. (Eds.): ICCSA 2015, Part II, LNCS 9156, pp. 534–548, 2015.
DOI: 10.1007/978-3-319-21407-8_38

$E\,;\;\; g_j(x) \leq 0, j \in I\}$. Thus, the purpose of the maximization problem (1) is to find all local maximizers, *i.e.*, all points $x^* \in \mathcal{R}$ such that condition (2) holds:

$$\forall x \in V_\epsilon(x^*) \cap \mathcal{R},\;\; f(x^*) \geq f(x). \tag{2}$$

where $V_\epsilon(x^*)$ is a neighborhood of x^*, with a positive ray ϵ.

It is also assumed that problem (1) has a finite number of isolated global and local maximizers. The existence of multi-solutions (local and global) makes this problem a great challenge that may be tackled with parallel solving techniques.

Methods for solving multilocal optimization problems include evolutionary algorithms, such as genetic [1] and particle swarm [13] algorithms, and additional contributions, like [6,15,20,23,24]. Stretched Simulated Annealing (SSA) was also proposed [14], combining simulated annealing and a stretching function technique, to solve unconstrained multilocal programming problems.

In previous work [16,18], Parallel Stretched Simulated Annealing (PSSA) was introduced as a parallel version of SSA, based on the decomposition of the feasible region in several subregions to which SSA is independently applied by a set of processors. Several domain decomposition and distribution approaches were explored, leading to successively increasing levels of numerical efficiency.

More recently, the parallel solving of constrained multilocal programming problems was also proposed, through coPSSA [19] (constrained PSSA), that couples the penalty method with PSSA. Basically, coPSSA creates a homogeneous partition of the iteration set of the l_1 penalty function [11]; each iteration subset is run in parallel, by different processors of a shared memory system, and each specific iteration invokes PSSA; this, in turn, involves additional processors; these processors are usually from a distributed memory cluster, but may also be from the same shared memory host, once PSSA is a MPI-based application [8].

In this paper, coPSSA is explored, with the l_1 penalty function, to solve four well known test problems [4], in order to analyze the kind of performance gains that may be expected with a reasonable set of parallel configurations. Moreover, the effect of the variation of the reduction factor parameter τ of the l_1 function is analyzed to investigate the existence of values leading to faster convergence.

The rest of the paper is organized as follows. Section 2 revises the basic ideas behind SSA and PSSA. Section 3 covers the basics of penalty method with l_1 penalty function, and provides some details on coPSSA design and implementation. Section 4 presents performance and numerical results from the evaluation of coPSSA. Finally, Section 5 concludes and defines directions for future work.

2 Unconstrained Optimization

2.1 Stretched Simulated Annealing

Stretched Simulated Annealing (SSA) is a multilocal programming method that solves bound constrained optimization problems. These may be described as:

$$\max_{x \in X} \varphi(x), \tag{3}$$

where $\varphi : \mathbb{R}^n \to \mathbb{R}$ is a given n-dimensional multimodal objective function and X is the feasible region defined by $X = \{x \in \mathbb{R}^n : -l_i \leq x_i \leq l_i,\ i = 1, ..., n\}$.

SSA solves a sequence of global optimization problems in order to compute the local solutions of the maximization problem (3). The objective function of each global problem is generated using a stretching function technique [12].

Let x^* be a solution of problem (3). The mathematical formulation of the global optimization problem is as follows:

$$\max_{x \in X} \varPhi_l(x) \equiv \begin{cases} \hat{\phi}(x) \text{ if } x \in V_\varepsilon(x^*) \\ \varphi(x) \text{ otherwise} \end{cases} \tag{4}$$

where $V_\varepsilon(x^*)$ is the neighborhood of solution x^* with a ray $\varepsilon > 0$.

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

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

where δ_1, δ_2 and κ are positive constants, and $\bar{\phi}(x)$ is

$$\bar{\phi}(x) = \varphi(x) - \frac{\delta_1}{2}\|x - x^*\|[\text{sign}(\varphi(x^*) - \varphi(x)) + 1]. \tag{6}$$

To solve the global optimization problem (4) the Simulated Annealing (SA) method is used [5]. The Stretched Simulated Annealing algorithm stops when no new optimum is identified after r consecutive runs. [15,16] provide more details.

2.2 Parallel Stretched Simulated Annealing

As a parallel implementation of the SSA method, PSSA was thoroughly described in previous work [16]. In this section, only a brief description is provided.

SSA applies a stochastic algorithm, ι successive times, in the feasible region of the bound constrained problem. The same algorithm may be applied to the subregions of a partition of the feasible region, with the aim of improving the number of optima found. Moreover, each subregion may be processed independently, once there are no data or functional dependencies involved. SSA is thus an embarrassingly parallel problem, calling for a Domain Decomposition approach.

In this regard, PSSA supports both homogeneous and heterogeneous decomposition of the feasible region, as well as static or dynamic assignment of subregions to processors. The various combinations of these possibilities are materialized in three PSSA variants: i) PSSA-HoS (Homogeneous decomposition, Static assignment), where subregions are defined only once, have equal size and processors self assign the same number of subregions; ii) PSSA-HoD (Homogeneous decomposition, Dynamic assignment), different from PSSA-HoS only in subregions being assigned to processors on-request (thus possibly in varying number); iii) PSSA-HeD (Heterogeneous decomposition, Dynamic assignment), based on an adaptive recursive refinement of an initial homogeneous partition of the feasible region, leading to an unpredictable number of subregions, of variable size, dynamically generated and processed on-demand, until certain stop criteria are met. PSSA-HoS and PSSA-HoD have the same numerical efficiency (*i.e.*, both find the same number of optima), but PSSA-HoD is faster due to its workload auto-balancing. PSSA-HeS usually finds more optima, but is also the slowest of the PSSA variants, once it typically searches many more subregions.

PSSA was written in C, runs on Linux systems, and it builds on MPI [8] (thus following the message passing paradigm). It is a SPMD (Single Program, Multiple Data) application that operates in a *master-slaves* configuration (*slave* tasks run SSA in subregions, under coordination of a *master* task), and may be deployed in shared-memory systems and/or in distributed-memory clusters.

3 Constrained Optimization

3.1 l_1 Penalty Method

There are three main classes of methods to solve constrained optimization problems [9,25]: 1) methods that use penalty functions, 2) methods based on biasing feasible over infeasible solutions, and 3) methods that rely on multi-objective optimization concepts. In this work constraints are handled using a class 1 method with the l_1 penalty function. This function is a classic penalty [11] defined by

$$\varphi(x,\mu) = f(x) - \frac{1}{\mu}\left[\sum_{k\in E}|h_k(x)| + \sum_{j\in I}[g_j(x)]^+\right]$$

where μ is a positive penalty parameter that progressively decreases to zero, along k_{max} iterations. A lower bound $\mu_{\min}$ is defined and μ is updated as follows:

$$\mu^{k+1} = \max\left\{\tau\mu^k, \mu_{\min}\right\} \tag{7}$$

where $k \in \{1,...,k_{max}\}$ represents the iteration, $\mu_{\min} \approx 0$ and $0 < \tau < 1$.

To solve the constrained problem (1), the penalty method solves a sequence of bound constrained problems, based on the l_1 penalty function, as defined by

$$\max_{x\in X} \varphi(x,\mu^k). \tag{8}$$

Problem (8) is solvable using PSSA. It is possible to prove that the solutions sequence $\{x^*(\mu^k)\}$ from (8) converges to the solution x^* of problem (1) [11,15].

The penalty method stops when a maximum number of iterations (k_{max}) is reached, or successive solutions are similar, accordingly with the next criteria:

$$\left|f(x^k) - f(x^{k-1})\right| \leq \epsilon_1 \quad \wedge \quad \left\|x^k - x^{k-1}\right\| \leq \epsilon_2 \tag{9}$$

where k is a given iteration of the penalty method.

The optimum value for the reduction factor τ of the update expression (7) is an open research issue in the optimization field [10,21]. This paper presents an analysis of the effect of the variation of τ in the l_1 penalty function method, when solving multilocal programming problems by PSSA. The main goal is to identify the best value τ in order to obtain the optima set in the shortest time.

3.2 Constrained PSSA

Constrained PSSA (coPSSA) was already introduced in [19], albeit in the context of a simpler evaluation scenario than the one explored in this paper. Here coPSSA is revised, and some clarification is provided on its design and implementation.

As stated in the previous section, problem (8) may be solved using SSA, including its parallel implementations, like PSSA. However, coupling PSSA with a serial implementation of the penalty method is of limited benefit performance-wise: in the end, some extra optima may be detected (due to the extra efforts of PSSA), but the penalty method will not run faster. A possible approach to increase the performance of the penalty method is to parallelize its execution.

As it happens, the parallelization of the penalty method is trivial: this method will execute a certain maximum number of times or iterations, as defined by the parameter k_{max}; each iteration uses successively lower values of the penalty parameter μ; these values are completely deterministic, as given by (7); therefore, a partition of the iteration space $\{1, ..., k_{max}\}$ may be defined, a priori, such that mutually disjoint subintervals of this space are assigned to different processors or search tasks (with one task per processor); each task will then run the penalty method along its iteration subspace; as soon as a task reaches convergence (in accordance to criteria (9)), the others will stop the search (before moving on to its next iteration, a task checks if other has already converged to the solutions).

Having different processors/tasks starting the optima search with different values of μ, scattered along the interval of possible values $\{\mu^1, ..., \mu^{k_{max}}\}$, may allow to reach convergence sooner, as compared to a single full sequential scan of that interval. However, it was noted that this strategy does not necessarily pay off; it must be tested for each and different constrained optimization problem, once the value(s) of μ that lead to convergence are unpredictable by nature.

coPSSA adopts the strategy above, for the parallelization of the penalty method, by performing a homogeneous decomposition of the iteration space: given P search tasks and k_{max} iterations, any search task t_p (with $p = 1, ..., P$) will iterate through approximately the same number of iterations, $w = \lceil \frac{k_{max}}{P})\rceil$, where $\lceil x \rceil$ is the smallest integer value not less than x; the iteration subinterval for a task t_p is then $\{k^p_{left}, ..., k^p_{right}\}$, where $k^p_{left} = ((p-1) \times w) + 1$ and $k^p_{right} = p \times w$; when a uniform width w is not possible ($k_{max}\ mod\ P \neq 0$), coPSSA makes the necessary adjustments in the last iteration subinterval (the one for task p); for instance, with $k_{max} = 100$ and $P = 2$, it is obtained $w = \lceil 50.0 \rceil = 50$ and so t_1 will iterate through $\{1, ..., 50\}$ and t_2 will iterate through $\{51, ..., 100\}$; with $P = 3$ it is obtained $w = \lceil 33.3(3) \rceil = 34$, and so t_1, t_2 and t_3 will iterate through $\{1, ..., 34\}$, $\{35, ..., 68\}$ and $\{69, ..., 100\}$, respectively.

Like PSSA, coPSSA was also coded in C, for performance reasons, and also to reuse previous code from one of the authors. In coPSSA, search tasks are conventional UNIX processes, that synchronize and exchange data using classical System V IPC mechanisms, like semaphores and shared memory[1]. When coPSSA

[1] Alternatives like Pthreads, OpenMP or even MPI, may be explored in the future.

starts, it forks P child processes (one per search task), where P is usually defined to be the number of CPU-cores of the host running coPSSA. Each child (search task) will then run the penalty method through a specific iteration subinterval; for each particular iteration, PSSA must be invoked; this is simply accomplished using the `system` primitive, that spawns second-level childs to run the `mpiexec` command[2]; in turn, this command runs PSSA, in a MPI master-slaves configuration, in a set of hosts / CPU-cores that are defined by coPSSA; basically, coPSSA is supplied with a base MPI "machinefile", from which extracts the computing resources to be assigned to each PSSA execution; the results produced by PSSA executions are written in specific files, which are then checked by the search tasks, in order to detect a possible convergence.

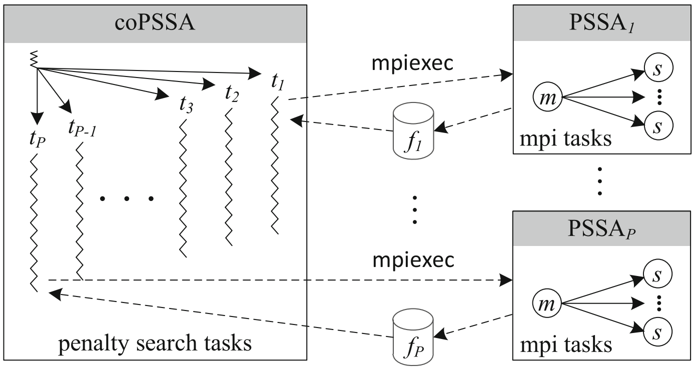


Fig. 1. The coPSSA application and its interactions with PSSA (t_p are coPSSA search tasks; m and s are PSSA tasks (master and slaves); f_p are result files from PSSA)

Figure 1 is a representation of coPSSA, including its interactions with PSSA. coPSSA may thus be viewed as a hybrid application, in the sense that combines its internal usage of shared-memory based parallelism, with an external component (PSSA) specially suited to exploit distributed-memory parallelism.

4 Evaluation

4.1 Setup

The experimental evaluation performed in the context of this work was carried in a commodity cluster of 9 hosts (1 frontend, and 8 worker nodes), with one Intel Core-i7 4790K 4.0GHz quad-core CPU per node, under Linux ROCKS version 6.1.1, with the Gnu C Compiler (GCC) version 4.4.7 and OpenMPI version 1.5.4.

coPSSA was always executed in the cluster frontend, with the number of search tasks (P) ranging from 1 to 8 (no overload was observed for $P > 4$,

[2] As a side note, we had to switch the MPI implementation, from MPICH2 to OpenMPI, because a bug in MPICH2 corrupts *stdio* in coPSSA after `mpiexec` returns.

despite the frontend having only a 4-core CPU). PSSA executions took place in the 8 worker nodes; these offer a total of 32 CPU-cores, fully specified in the base MPI hostfile supplied to coPSSA, that are used, four at a time, to service the PSSA execution requests of each coPSSA task; thus, each PSSA execution always consumed 4 cores, with 1 core for the master process, and 3 cores for slave processes; in order to fully utilize these 3 cores, the number of subregions for each PSSA execution was set to be as close as possible (in excess) of 3; for 2-dimensional problems, like the ones tested, this implies 4 subregions [16]. The PSSA variant used throughout the tests was always PSSA-HoD: not only it is appropriate for a fixed number of subregions, but is also the fastest variant.

All tests shared the penalty algorithm parameters k_{max}=100, μ^0=1.0, $\mu_{min} = 10^{-6}$ and $\tau \in \{0.1, ..., 0.9\}$, and the convergence parameters $\epsilon_1 = \epsilon_2 = 10^{-4}$. The PSSA numerical parameters were $r = 5$, $\delta_1 = 1.5$, $\delta_2 = 0.5$ and $\kappa = 0.05$.

4.2 Search Times

Tables 4 to 7 show the search times measured for the selected benchmark problems, for all valid combinations of P (number of coPSSA search tasks) and τ (the penalty reduction factor under evaluation), with $P = 1, 2, ..., 8$ and $\tau = 0.1, 0.2, ..., 0.9$. The search times are the times required by coPSSA to converge to the constrained problem optima. Figures 2 to 5 represent the data of the tables, with four subfigures per table, that offer four different and complementary perspectives on search times (the first two – (a) and (b) – build on a horizontal reading of the table, and the last two – (c) and (d) – build on a vertical reading):

(a) "Search times for each τ": one curve per τ value, based on the search times obtained with a fixed τ, when varying the number P of search tasks; allows to verify the influence of the variation of the degree of search parallelism (P) on the search times produced by a specific τ; allows also to identify the τ value that ensures the lowest (absolute minimum) search times;
(b) "Average search times for $P \geq 1$ and $P \geq 2$": one curve, where each point is the average of the search times obtained with a fixed τ and all possible values of P (*i.e.*, $P \geq 1$); another curve, where each point is the average of the search times obtained with a fixed τ and all values of $P \geq 2$; the first curve allows to deduce the best τ, irregardless of the use of sequential ($P = 1$) or parallel ($P \geq 2$) searches; the second curve allows to conclude which τ value is the best when only parallel searches are made;
(c) "Search times for each P": one curve per P value, based on the search times obtained with a fixed P, when varying τ; allows to verify the influence of the variation of the value of τ on the search times produced by a specific P; allows also to identify the P value that ensures the lowest search times;
(d) "Average search times for all τ values": a single curve, where each point is the average of the search times obtained with a fixed P and all possible values of τ; allows to deduce the best P, irregardless of the value of τ.

For each of the above perspectives, these are the main conclusions that may be derived from the experimental data[3]:

(a) in all problems, when increasing the number of search tasks, the trend followed by the search times for each τ is mostly downwards, eventually followed by stabilization; this trend is more regular for Problems G6 and G8 (see Figures 2a and 3a), slightly less regular in Problem G12 (see Figure 5a), and much more irregular in Problem G11 (see Figure 4a); it is also possible to identify the values of τ that attain the absolute minimum search times; these times are boxed (within a tolerance of 5%), in Tables 4 to 7; the corresponding values of τ are gathered in the following table:

Table 1. Values of τ that ensure the absolute minimum search times

Problem	$P = 1$	$P \geq 2$
G6	0.1	0.5 , 0.6
G8	0.1	0.7
G11	0.3	0.9
G12	0.1	0.6 to 0.9

the previous table allows to conclude that when only sequential searches are used ($P = 1$), the absolute minimum values for search times are attained with small values of τ; however, if only parallel searches are conducted ($P \geq 2$), the lowest search times are achieved with higher (mid range to maximum) values of τ; one should note, though, that these observations cannot be generalized, because the τ values that produce the absolute minimum search times may not be the ones that produces the lowest average search times;

(b) the values of τ that ensure the lowest average search times depend on the specific problem, and also depend on whether both sequential or parallel searches are admitted ($P \geq 1$), or only parallel searches ($P \geq 2$) are allowed; in Tables 4 to 7, the columns "*avg.* $P \geq 1$" and "*avg.* $P \geq 2$" show the average search times attained by each τ, when $P \geq 1$ and $P \geq 2$, respectively; in each column, the lowest average search time is in bold (within a tolerance of 5%); the related values of τ are gathered in the following table:

Table 2. Values of τ that ensure the lowest average search times

Problem	$P \geq 1$	$P \geq 2$
G6	0.1	0.1 to 0.6
G8	0.1	0.8 , 0.9
G11	0.3	0.3
G12	0.1	0.1 to 0.6

the previous table allows to conclude that if using a sequential or a parallel search is indifferent ($P \geq 1$), then single small values of τ are best in order

[3] Note: for Problem G6, no values are shown for $\tau = 0.9$, because no convergence was reached within the maximum of $k_{max} = 100$ iterations.

to attain the lowest search times, on average (coincidentally, such values of τ match the values in Table 1 that ensure the absolute minimum search times for $P = 1$); however, if only parallel searches are admissible ($P \geq 2$), there may be several best values of τ to chose from, like in problems G6 and G12, that share the same range of values (0.1 to 0.6), or like in problem G8, with another range (0.8 to 0.9); problem G11, though, is an exception, because the best τ value is always the same (0.3) for sequential and parallel searches;

(c) when increasing τ, problem G6 (Figure 2c) and problem G12 (Figure 5c) exhibit a similar trend on the search times for each P (these times progressively increase for $P = 1$, but for $P \geq 2$ they are substantially lower and stable, and only start to increase for higher values of τ); for problem G8 (Figure 3c), the trend is also upwards for $P = 1$, but now is reversal for $P \geq 2$ (as τ values increase, the search times will decay); finally, for problem G11 (Figure 4c), the pattern is very irregular, independently of P (increasing τ may lead to an unpredictable surge or decline of search times);

(d) irregardless of τ, the use of parallel searches typically pays off for all problems (in general, increasing the level of parallelism (P) will decrease the search times); this downwards trend is more regular in problems G8 (Figure 3d), G11 (Figure 4d), and G12 (Figure 5d), where the lowest average search times are achieved with the higher values of P (values 7 and/or 8); in problem G6 (Figure 2d), both $P = 4$ and $P = 7$ provide the best average search times; all these times are shown encircled in the tables (within a tolerance of 5%).

4.3 Number of Optima

Another important issue is the number of optima found. Table 3 shows that number as an average: for each problem, sequential only ($P = 1$) and parallel only ($P \geq 2$) searches are considered; for each of these categories, it is presented the mean of the number of optima found with all values of τ tested (0.1,...,0.9); along with the mean, the coefficient of variation is also presented, in parenthesis.

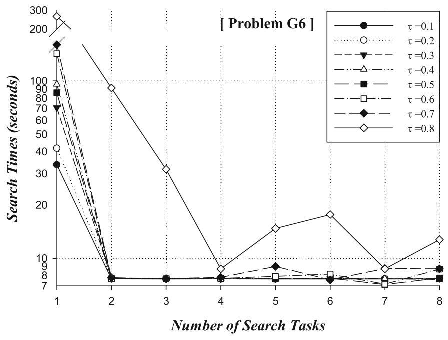
Table 3. Average number of optima found (and coefficient of variation)

Problem	$P = 1$	$P \geq 2$
G6	4.9 (20.3%)	5.7 (12.7%)
G8	6.0 (0%)	6.0 (0%)
G11	11.4 (21.5%)	11.9 (18.4%)
G12	10.8 (7.7%)	10.2 (6.9%)

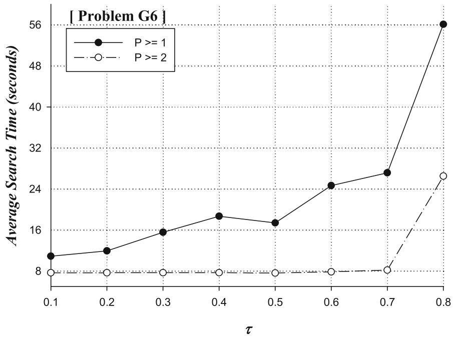
The table allows to conclude that: i) for problem G8, the number of optima found is the same, independently of P and τ; ii) for the other problems, there are indeed differences between using a single sequential search and using parallel searches; but these differences are small, and they may either favor parallel searches (problems G6 and G11) or sequential searches (problem G12); it seems, however, that parallel searches exhibit a smaller dispersion of the optima number.

Table 4. Search times values for Problem G6 (seconds)

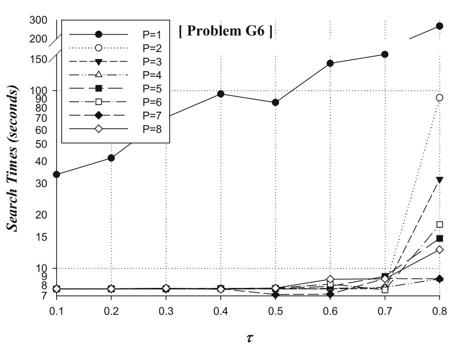
	P								avg.	avg.
τ	1	2	3	4	5	6	7	8	$P \geq 1$	$P \geq 2$
0.1	$\boxed{33.7}$	7.7	7.7	7.7	7.7	7.7	7.7	7.7	**10.9**	**7.7**
0.2	41.8	7.7	7.7	7.7	7.7	7.7	7.7	7.7	11.9	**7.7**
0.3	70.7	7.7	7.7	7.7	7.7	7.7	7.7	7.7	15.6	**7.7**
0.4	95.8	7.7	7.7	7.7	7.7	7.7	7.7	7.6	18.7	**7.7**
0.5	85.8	7.7	7.7	7.7	7.7	7.7	$\boxed{7.1}$	7.8	17.4	**7.6**
0.6	142.4	7.7	7.7	7.7	7.9	8.2	$\boxed{7.2}$	8.7	24.7	**7.8**
0.7	159.9	7.8	7.7	7.8	9.0	7.6	8.8	8.7	27.1	8.2
0.8	263.2	91.3	31.7	8.7	14.8	17.6	8.7	12.7	56.10	26.5
0.9										
avg.	111.7	18.1	10.7	7.8	8.8	9.0	7.8	8.6		



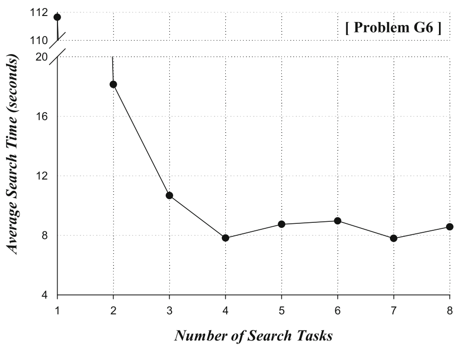
(a) Search times for each τ



(b) Average search times for $P \geq 1$ and $P \geq 2$



(c) Search times for each P

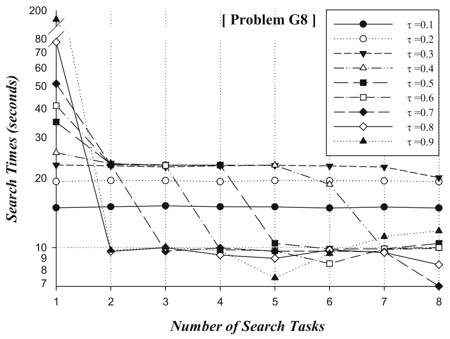


(d) Average search times for all τ values

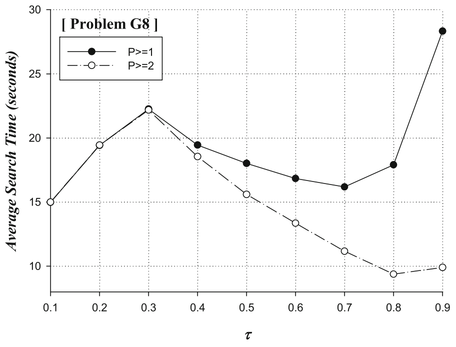
Fig. 2. Search times plots for Problem G6

Table 5. Search times values for Problem G8 (seconds)

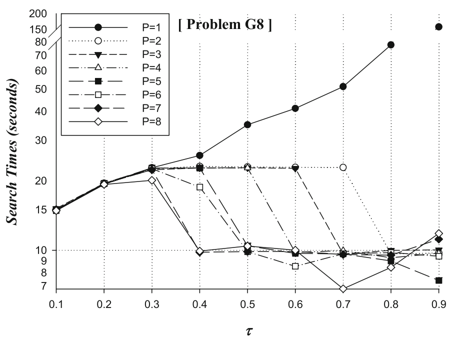
τ	P 1	2	3	4	5	6	7	8	avg. $P \geq 1$	avg. $P \geq 2$
0.1	$\boxed{14.9}$	15.0	15.2	15.0	15.0	14.9	15.0	14.9	**15.0**	15.0
0.2	19.3	19.5	19.5	19.3	19.5	19.5	19.5	19.3	19.4	19.4
0.3	22.8	22.6	22.3	22.6	22.7	22.6	22.3	20.1	22.3	22.2
0.4	25.8	23.1	22.8	22.8	22.7	18.8	9.8	9.9	19.5	18.6
0.5	35.0	23.0	22.8	22.8	10.5	9.9	9.9	10.4	18.0	15.6
0.6	41.2	22.9	22.7	9.8	9.7	8.5	9.9	10.0	16.8	13.4
0.7	51.2	22.9	9.7	10.0	9.7	9.7	9.6	$\boxed{6.8}$	16.2	11.2
0.8	77.6	9.6	10.0	9.3	9.0	9.8	9.5	8.4	17.9	**9.4**
0.9	157.3	9.7	10.0	9.7	7.4	9.4	11.2	11.8	28.3	**9.9**
avg.	49.5	18.7	17.2	15.7	14.0	13.7	12.9	13.0		



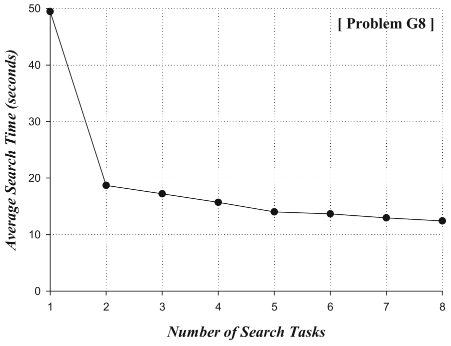
(a) Search times for each τ



(b) Average search times for $P \geq 1$ and $P \geq 2$



(c) Search times for each P

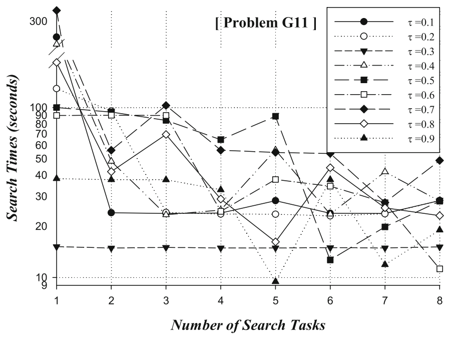


(d) Average search times for all τ values

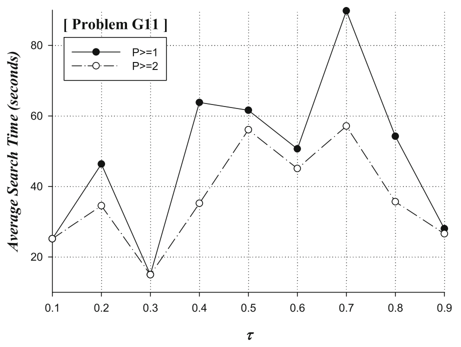
Fig. 3. Search times plots for Problem G8

Table 6. Search times values for Problem G11 (seconds)

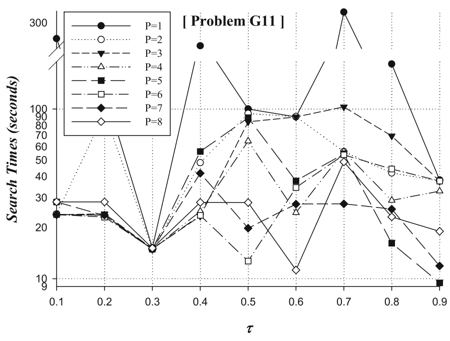
τ	P 1	2	3	4	5	6	7	8	avg. $P \geq 1$	avg. $P \geq 2$
0.1	274.8	24.0	23.8	24.0	28.3	23.9	23.8	28.4	56.4	25.2
0.2	129.2	95.4	24.1	23.6	23.6	23.0	23.7	28.3	46.4	34.5
0.3	$\boxed{15.2}$	14.9	15.0	14.9	14.9	15.0	14.9	15.1	**15.0**	**15.0**
0.4	264.1	48.3	23.4	24.9	56.2	23.6	41.8	28.2	63.8	35.2
0.5	100.2	94.1	84.1	64.8	88.9	12.7	19.8	28.1	61.6	56.1
0.6	89.9	90.2	89.9	24.5	37.7	34.4	27.5	11.3	50.7	45.1
0.7	318.3	56.2	103.0	56.2	54.5	53.8	27.6	48.8	89.8	57.1
0.8	184.2	41.9	69.5	28.9	16.2	44.3	25.7	23.1	54.2	35.7
0.9	38.1	37.6	37.6	32.8	$\boxed{9.5}$	37.6	11.9	19.0	28.0	26.6
avg.	157.1	55.9	52.3	32.7	36.6	29.8	$\enclose{circle}{24.1}$	25.6		



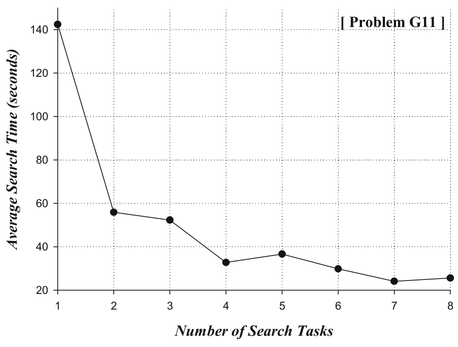
(a) Search times for each τ.



(b) Average search times for $P \geq 1$ and $P \geq 2$



(c) Search times for each P

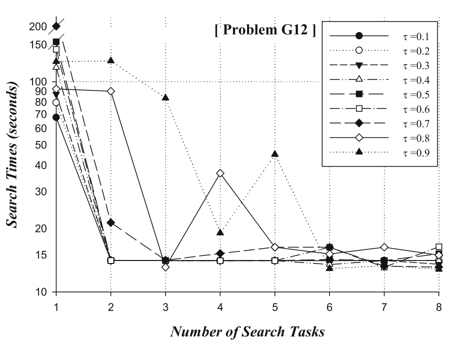


(d) Average search times for all τ values

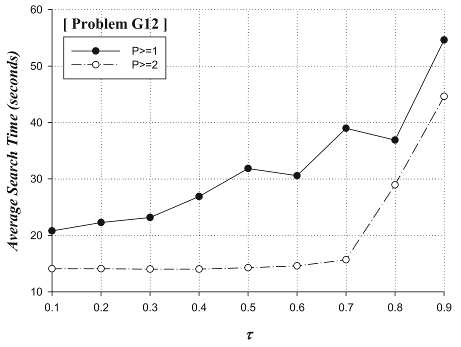
Fig. 4. Search times plots for Problem G11

Table 7. Search times values for Problem G12 (seconds)

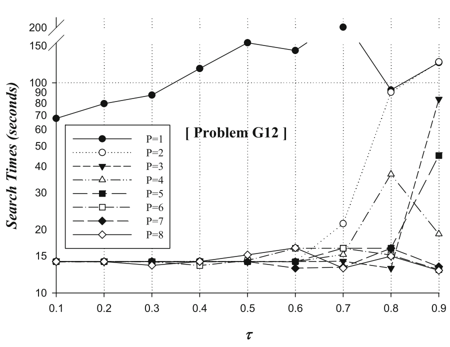
τ	P 1	2	3	4	5	6	7	8	avg. $P \geq 1$	avg. $P \geq 2$
0.1	$\boxed{67.6}$	14.1	14.1	14.1	14.1	14.1	14.1	14.1	**20.8**	**14.1**
0.2	79.5	14.1	14.1	14.1	14.1	14.1	14.1	14.1	22.3	**14.1**
0.3	87.3	14.1	14.1	14.1	14.1	14.1	14.1	13.5	23.2	**14.0**
0.4	116.9	14.2	14.1	14.1	14.1	13.5	14.1	14.1	26.9	**14.0**
0.5	154.9	14.1	14.1	14.1	14.1	14.3	14.1	15.2	31.8	**14.3**
0.6	142.4	14.1	14.1	14.1	14.1	16.3	$\boxed{13.1}$	16.4	30.6	**14.6**
0.7	202.1	21.4	14.2	15.2	16.3	16.3	$\boxed{13.3}$	$\boxed{13.1}$	39.0	15.7
0.8	92.4	90.1	$\boxed{13.1}$	36.7	16.3	15.1	16.3	15.0	36.9	29.0
0.9	124.7	125.6	83.6	19.1	45.1	$\boxed{12.9}$	$\boxed{13.3}$	$\boxed{12.8}$	54.6	44.6
avg.	118.7	35.8	21.7	17.3	18.0	14.5	14.1	14.3		



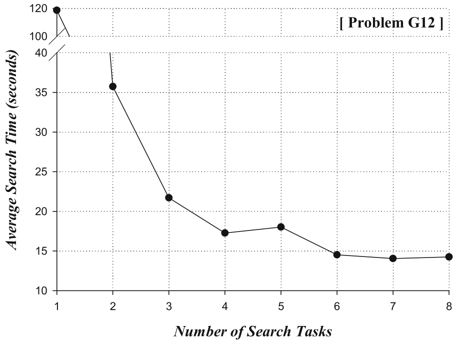
(a) Search times for each τ



(b) Average search times for $P \geq 1$ and $P \geq 2$



(c) Search times for each P



(d) Average search times for all τ values

Fig. 5. Search times plots for Problem G12

5 Conclusions and Future Work

In this paper coPSSA is explored, as a hybrid application that solves constrained optimization problems, by integrating a numerical l_1 penalty method with a parallel solver of unconstrained (bound constrained) problems (PSSA).

The effect of the variation of the number of search tasks, and of the penalty parameter reduction factor (τ) was studied, in the context of the l_1 penalty function. With base on the analysis of the results obtained with the four tested problems, it is possible to conclude: i) increasing the number of search tasks typically decreases the search times; ii) for the tested problems, smaller values of τ typically imply lower average search times; iii) for some problems, the number of optima found does not depend on the number of search tasks neither on the value of τ, while other problems are sensitive to the variation of those factors.

In the future, the research team intends to refine this work, by solving more constrained problems (including problems with more than 2 dimensions), and exploring higher levels of parallelism (*i.e.*, by running coPSSA with $P >> 8$).

Acknowledgments. This work was been supported by FCT (Fundação para a Ciência e Tecnologia) in the scope of the project UID/CEC/00319/2013.

References

1. Chelouah, R., Siarry, P.: A continuous genetic algorithm designed for the global optimization of multimodal functions. Journal of Heuristics **6**, 191–213 (2000)
2. Eriksson, P., Arora, J.: A comparison of global optimization algorithms applied to a ride comfort optimization problem. Structural and Multidisciplinary Optimization **24**, 157–167 (2002)
3. Floudas, C.: Recent advances in global optimization for process synthesis, design and control: enclosure of all solutions. Computers and Chemical Engineering, 963–973 (1999)
4. Hedar, A.-R.: Global Optimization Test Problems. http://www-optima.amp.i.kyoto-u.ac.jp/member/student/hedar/Hedar_files/TestGO.htm
5. Ingber, L.: Very fast simulated re-annealing. Mathematical and Computer Modelling **12**, 967–973 (1989)
6. Kiseleva, E., Stepanchuk, T.: 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)
7. León, T., Sanmatías, S., Vercher, H.: A multi-local optimization algorithm. Top **6**(1), 1–18 (1998)
8. Message Passing Interface Forum. http://www.mpi-forum.org/
9. Michalewicz, Z.: A survey of constraint handling techniques in evolutionary computation methods. In: Proceedings of the 4th Annual Conference on Evolutionary Programming, pp. 135–155 (1995)
10. Mongeau, M., Sartenaer, A.: Automatic decrease of the penalty parameter in exact penalty function methods. European Journal of Operational Research **83**, 686–699 (1995)
11. Nocedal, J., Wright, S.: Numerical Optimization. Springer Series in Operations Research. Springer (1999)

12. Parsopoulos, K., Plagianakos, V., Magoulas, G., Vrahatis, M.: Objective function stretching to alleviate convergence to local minima. Nonlinear Analysis **47**, 3419–3424 (2001)
13. Parsopoulos, K., Vrahatis, M.: Recent approaches to global optimization problems through particle swarm optimization. Natural Computing **1**, 235–306 (2002)
14. Pereira, A.I., Fernandes, E.M.G.P.: Constrained multi-global optimization using a penalty stretched simulated annealing framework. In: AIP Conference Proceedings Numerical Analysis and Applied Mathematics, vol. 1168, pp. 1354–1357 (2009)
15. Pereira, A.I., Ferreira, O., Pinho, S.P., Fernandes, E.M.G.P.: Multilocal programming and applications. In: Zelinka, I., Snasel, V., Abraham, A. (eds.) Handbook of Optimization. ISRL, vol. 38, pp. 157–186. Springer, Heidelberg (2013)
16. Pereira, A.I., Rufino, J.: Solving multilocal optimization problems with a recursive parallel search of the feasible region. In: Murgante, B., Misra, S., Rocha, A.M.A.C., Torre, C., Rocha, J.G., Falcão, M.I., Taniar, D., Apduhan, B.O., Gervasi, O. (eds.) ICCSA 2014, Part II. LNCS, vol. 8580, pp. 154–168. Springer, Heidelberg (2014)
17. Price, C.J., Coope, I.D.: Numerical experiments in semi-infinite programming. Computational Optimization and Applications **6**, 169–189 (1996)
18. Ribeiro, T., Rufino, J., Pereira, A.I.: PSSA: parallel stretched simulated annealing. In: AIP Conference Proceedings, Numerical Analysis and Applied Mathematics, vol. 1389, pp. 783–786 (2011)
19. Rufino, J., Pereira, A.I., Pidanic, J.: coPSSA - Constrained parallel stretched simulated annealing. In: Proceedings of the 25th Int. Conference Radioelektronika 2015, pp. 435–439 (2015)
20. Salhi, S., Queen, N.: A Hybrid Algorithm for Identifying Global and Local Minima When Optimizing Functions with Many Minima. European Journal of Operations Research **155**, 51–67 (2004)
21. Shandiz, R.A., Tohidi, E.: Decrease of the Penalty Parameter in Differentiable Penalty Function Methods. Theoretical Economics Letters **1**, 8–14 (2011)
22. Surjanovic, S., Bingham, D.: Virtual Library of Simulation Experiments: Test Functions and Datasets. http://www.sfu.ca/ssurjano
23. Tsoulos, I., Lagaris, I.: Gradient-controlled, typical-distance clustering for global optimization. http://www.optimization.org (2004)
24. Tu, W., Mayne, R.: Studies of multi-start clustering for global optimization. International Journal Numerical Methods in Engineering **53**, 2239–2252 (2002)
25. Wang, Y., Cai, Z., Zhou, Y., Fan, Z.: Constrained optimization based on hybrid evolutionary algorithm and adaptive constraint-handling technique. Structural and Multidisciplinary Optimization **37**, 395–413 (2008)