Numer Algor (2012) 60:613–629
DOI 10.1007/s11075-012-9590-x

ORIGINAL PAPER

Properties and numerical testing of a parallel global optimization algorithm

Marco Gaviano · Daniela Lera

Received: 2 December 2011 / Accepted: 9 May 2012 /
Published online: 24 May 2012

Abstract In the framework of multistart and local search algorithms that find the global minimum of a real function $f(x)$, $x \in S \subseteq R^n$, Gaviano *et alias* proposed a rule for deciding, as soon as a local minimum has been found, whether to perform or not a new local minimization. That rule was designed to minimize the average local computational cost $eval_1(\cdot)$ required to move from the current local minimum to a new one. In this paper the expression of the cost $eval_2(\cdot)$ of the entire process of getting a global minimum is found and investigated; it is shown that $eval_2(\cdot)$ has among its components $eval_1(\cdot)$ and can be only monotonically increasing or decreasing; that is, it exhibits the same property of $eval_1(\cdot)$. Moreover, a counterexample is given that shows that the optimal values given by $eval_1(\cdot)$ and $eval_2(\cdot)$ might not agree. Further, computational experiments of a parallel algorithm that uses the above rule are carried out in a MatLab environment.

Keywords Random search · Global optimization · Parallel computing

1 Introduction

Global optimization problems of the following type

Problem 1.1

$$find \;\; x^* \in S, \;\; such\ that \;\; f(x^*) \leq f(x), \;\; \forall x \in S,$$

M. Gaviano · D. Lera (✉)
Dipartimento di Matematica e Informatica,
University of Cagliari, Cagliari, Italy
e-mail: lera@unica.it

where $f : S \to \mathbf{R}$ is a function defined on a set $S \subseteq \mathbf{R}^n$, are often encountered in the mathematical representation of real-world problems. While there do exist very efficient algorithms to find a local minimum of the object function in Problem 1.1, the search of a global minimum can be a very hard problem. Indeed, Nemirovsky and Yudin [14], and Vavasis [19] have proved, under suitable assumptions, that the computational complexity of the global optimization problem is exponential.

Numerical techniques for finding solutions to such problems by using parallel schemes have been discussed in the literature (see, e.g., [4, 9, 17, 18, 20]. Many procedures introduced in the literature to solve Problem 1.1 employ local minimum algorithms; these mostly construct, by starting from a point x_0, a sequence $\{x_j\}$, with $f(x_j) > f(x_{j+1})$, that converges to a local minimum x^*. The starting point x_0 is chosen uniformly at random in S or according to a probability distribution based on the algorithm running time history. Clearly, if we start from different points in S, we can expect to find all the local minima of $f(\cdot)$ and then its global minimum. Researchers have proposed different strategies for selecting the starting points of the local searches; see the papers by Boender and Rinnooy Kan [1], Cetin et al. [2], Desai and Patil [3], Hedar and Fukushima [10] Levy and Montalvo [12], Lucidi and Piccioni [13], Oblow [15].

The main point to tackle whenever local search strategies are used to find a global minimum, is to avoid finding the same local minima. One can choose the starting point x_0 of a new local search such that the function value $f(x_0)$ is less than the value of the last local minimum found. In such a way the local searches guarantee that a new local minimum point with function value less than the previous ones can be found. On the other hand, by proceeding in this way, we reduce the size of the region that could take us to the global minimum. That is depicted in Fig. 1.

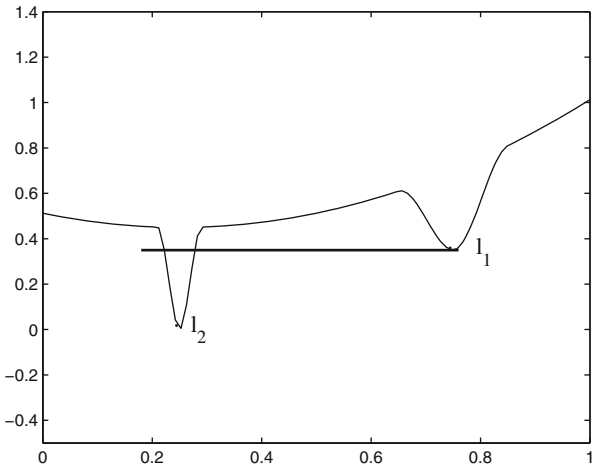

Fig. 1 Graph of a two local minima l_1 and l_2 function

It is clear that the size of the region of points that can take us to l_2 is far smaller once we have found l_1. If we assume the condition of choosing as a starting point of a local search a point with function value less than $f(l_1)$, it gets more difficult for the local search to lead to a global minimum.

In [8] an algorithm *Glob* was proposed that chooses uniformly at random in S a point x_0 and then, starting from it, according to an optimal rule a new local search is or is not carried out. This rule assumes that both the probabilities of getting any local minimum starting from an arbitrary $x_0 \in S$ and the required computational costs are known; further, it was derived so that the average computational cost is minimal. Since usually probabilities and cost are not known, in the implementation those parameters were approximated along the minimization process taking into account of the previous computational history.

In the present paper the properties of the optimal rule proposed in [8] are investigated. That rule was found by minimizing the computational cost $eval_1(\cdot)$ required to move from the current local minimum to a new one. Here we consider the computational cost $eval_2(\cdot)$ of the entire process of finding the global minimum; it is found that $eval_2(\cdot)$ exhibits the same general features as $eval_1(\cdot)$ and has among its components $eval_1(\cdot)$. Moreover, a counterexample is given that shows that the optimal values given by $eval_1(\cdot)$ and $eval_2(\cdot)$ cannot agree.

Further, numerical experiments are realized with the code written in the Matlab programming language and using the Matlab parallel toolboxes. The codes are executed on two computers equipped with four and six processors, respectively; fourteen configurations of the computing resources have been investigated. To evaluate the algorithm performances the *speedup* and the *efficiency* are reported for each configuration.

2 Preliminaries

In this section we state assumptions and definitions that will be used throughout the paper; further, we report the results established in [8] to which the new findings are related. For Problem 1.1 we consider the following assumption.

Assumption 2.1

(i) *$f(\cdot)$ has a finte number m of local minimum points l_i, $i = 1, ..., m$ and $f(l_i) > f(l_{i+1})$;*
(ii) *$meas(S) = 1$,*

with meas(S) denoting the measure of S.

Consider the following algorithm scheme.

Algorithm 2.1 (Algorithm Glob)

Choose x_0 uniformly on S;
$i \leftarrow 1;\ j \leftarrow 1;$
$(x_1,\ fx_1) \leftarrow local\_search(x_0);$
$l_i = x_1;\ fl_i = fx_1;$
repeat
$j \leftarrow j + 1;$
choose x_0 uniformly on S;
if $f(x_0) \leq fl_i$ or ($f(x_0) > fl_i$ and $rand(1) < d_i$)
$(x_1,\ fx_1) \leftarrow local\_search(x_0);$
if $fx_1 < fl_i$
$i \leftarrow i + 1;$
$l_i \leftarrow x_1;\ fl_i \leftarrow fx_1;$
end if
end if
until a stop rule is met;
end

The parameters d_i are probability values, that is $d_i \in [0, 1]$. The function $rand(1)$ denotes a generator of random numbers in the interval $[0, 1]$. Further, we denote by $local\_search(x_0)$ any procedure that starting from a point x_0 returns a local minimum l_i of Problem 1.1 and its function value.

In algorithm *Glob* a sequence of local searches is carried out. Once a local search has been completed and a new local minimum l_j is found, a point x_0 at random uniformly on S is chosen. Whenever $f(x_0)$ is less than $f(l_j)$ a new search is performed from x_0; otherwise a local search is performed with probability d_i. We assume that Problem 1.1 satisfies all the conditions required to make the procedure $local\_search(x_0)$ convergent. We have the following proposition.

Proposition 2.1 *Let Assumption* 2.1 *hold and consider a run of algorithm Glob. Then the probability that l_i is a global minimum of Problem* 1.1 *tends to one as $j \to \infty$.*

First, we settle the following notation.

Definition 2.1

- $A_{0,j} \equiv \{x \in S \mid$ starting from x, $local\_search(\cdot)$ returns local minimum $l_j\}$;
- $A_{i,j} \equiv \{x \in S \mid f(x) \leq f(l_i)$; starting from x, $local\_search(\cdot)$ returns local minimum $l_j\}$;
- $p_{0,j} = meas(A_{0,j})$;
- $p_{i,j} = meas(A_{i,j})$.

We have

$$\sum_{i=1}^{m} p_{0,i} = meas(S) = 1.$$

We consider the following definitions and assumptions for algorithm *Glob*.

Definition 2.2

- $t_i \equiv$ the probability that having found the local minimum l_i, in a subsequent iteration no new local minimum is detected;
- $tr(i_1, ..., i_p) \equiv$ the set (trajectory) of p local minimum points $l_{i_1}, ..., l_{i_p}$ found in a run of algorithm *Glob*;
- $Prob_{i,j}(d_i) \equiv$ the probability that algorithm *Glob* having found the local minimum l_i can find the local minimum l_j in a subsequent iteration;
- $Prob_{i,j}^{(\infty)}(d_i) \equiv$ the probability that algorithm *Glob* having found the local minimum l_i can find l_j assuming that an infinity number of iterations is carried out;
- $Prob_{tr}^{(n)}(d_{i_1}, ..., d_{i_{p-1}}) \equiv$ the probability that algorithm *Glob* constructs the trajectory $tr = (i_1, ..., i_p)$ in n iterations.

Assumption 2.2

- *algorithm Glob runs an infinite number of iterations;*
- *the number of function evaluations required by local_search is* $k = constant$.

In [8], the following theorem was proved.

Theorem 2.1 *The average number of function evaluations so that algorithm Glob, having found a local minimum* l_i, *finds any new one is given by*

$$evals_1(d_i) = f_i \frac{1}{Prob_{i,*}}, \quad i = 1, ..., m-1, \tag{2.1}$$

with

$$Prob_{i,*} = \sum_{j=i+1}^{m} p_{i,j} + d_i \left(\sum_{j=i+1}^{m} p_{0,j} - \sum_{j=i+1}^{m} p_{i,j} \right),$$

$$f_i = k \sum_{j=i+1}^{m} p_{i,j} + kd_i \left(1 - \sum_{j=i+1}^{m} p_{i,j} \right) + (1 - d_i) \left(1 - \sum_{j=i+1}^{m} p_{i,j} \right).$$

Further, in [8], the following problem was stated and investigated

Problem 2.1 *Let us consider Problem* 1.1 *and let the values k,* $p_{0,j}$ *and* $p_{i,j}$ *be given. Find a value* d_i^* *such that*

$$evals_1\left(d_i^*\right) = \min_{d_i} evals_1(d_i).$$

It turns out that the sign of the derivative of $evals_1(d_i)$ is constant in [0, 1] and is greater than or equal to zero if

$$k \geq \frac{\left(\sum_{j=i+1}^{m} p_{0,j}\right)\left(1 - \sum_{j=i+1}^{m} p_{i,j}\right)}{\left(\sum_{j=i+1}^{m} p_{i,j}\right)\left(1 - \sum_{j=i+1}^{m} p_{0,j}\right)}. \tag{2.2}$$

The latter links the probability $p_{i,j}$ with the number k of function evaluations performed at each local search in order to choose the most convenient value of d_i: if the condition is met, $d_i = 0$ is suitable to be chosen otherwise $d_i = 1$. That is,

$$d_i = \begin{cases} 0 \text{ if } k > (p_2 \cdot (1 - p_3))/(p_3 \cdot (1 - p_2)), \\ 1 \text{ otherwise,} \end{cases} \tag{2.3}$$

with

$$p_2 = \sum_{j=i+1}^{m} p_{0,j}, \;\; p_3 = \sum_{j=i+1}^{m} p_{i,j}.$$

In real problems usually the values $p_{0,j}$ and $p_{i,j}$ are not known; hence the choice of probabilities $d_1, d_2, ..., d_{m-1}$ in the optimization of the function in Problem 1.1 cannot be calculated exactly.

Algorithm 2.1 has been completed with d_i given by (2.3) and p_2, p_3 and k aproximated as follows

$$\begin{aligned} p_2 \quad & 1/(\text{number of searches carried out }), \\ p_3 = \; & 1/(\text{number of iterations already carried out}), \\ k = \; & \text{mean number of function evaluations carried out in each local search.} \end{aligned} \tag{2.4}$$

3 New results

In this section we assume that algorithm *Glob* can run an infinite number of iterations. We calculate the average number of function evaluation such that the global minimum point is found. Further, it is assumed that we know the values $p_{0,j}$ and $p_{i,j}$, $i = 1, ..., m-1$ and $j = 1, ..., m$, and that the number of function evaluations required by *local_search* is $k =$ constant.

We first prove two lemmas.

Lemma 3.1

$$t_i = \sum_{j=1}^{i} p_{0,j} + \left(\sum_{j=i+1}^{m} p_{0,j} - \sum_{j=i+1}^{m} p_{i,j} \right) (1 - d_i),$$

$$Prob_{i,j}(d_i) = p_{i,j} + d_i \left(p_{0,j} - p_{i,j} \right),$$

$$\begin{aligned} Prob_{tr}^{(n)}(d_1, \ldots, d_{p-1}) = {} & p_{0,i_1} \cdot \left(p_{i_1,i_2} + (p_{0,i_2} - p_{i_1,i_2}) d_{i_1} \right) \cdot \ldots \cdot \\ & \cdot \left(p_{i_{p-i},i_p} + (p_{0,i_p} - p_{i_{p-1},i_p}) d_{i_{p-1}} \right) \cdot \\ & \cdot \sum_{\substack{j_1,\ldots,j_{p-1}=0 \\ j_1+\ldots+j_{p-1} \le n-p}}^{n-p} t_{i_1}^{j_1} \cdot t_{i_2}^{j_2} \cdot \ldots \cdot t_{i_{p-1}}^{j_{p-1}}. \end{aligned}$$

with $n > p$ and $tr = (i_1, \ldots, i_p)$, $i_p = m$.

Proof We get the first statement by noting that whenever x_0 is chosen in $A_{0,j}$ as $j = 1, \ldots, i$, then no new local minimum can be found; while whenever x_0 is chosen in $A_{0,j} - A_{i,j}$, $j = i+1, \ldots, m$, then the probability of not moving from l_i is $(1 - d_i)$.

The second follows from this remark: we get a new local minimum both whenever the starting point x_0 is chosen in $A_{i,j}$ and, with probability, d_i, whenever is chosen in $A_{0,j} - A_{i,j}$. We prove the third statement by noting that in order to get the trajectory $t = (i_1, \ldots, i_p)$, we must first get the minimum in l_{i_1}; this takes place with probability p_{0,i_1}; next we must move from the local minimum l_{i_1} to the local minimum in l_{i_2} and so on until we get l_{i_p}. Further, we don't move j_1 times from l_{i_1}, j_2 times from l_{i_2},.. , and j_p times from l_{i_p}. Since $j_1 + j_2 + \ldots + j_p = n - p$, the lemma follows. □

Lemma 3.2

$$Prob_{i,j}^{(\infty)}(d_i) = \frac{p_{i,j} + d_i(p_{0,j} - p_{i,j})}{\sum_{l=i+1}^{m} (p_{i,l} + d_i(p_{0,l} - p_{i,l}))}, \tag{3.1}$$

$$Prob_{(i_1,\ldots,i_p)}^{(\infty)}(d_1, \ldots, d_{p-1}), = p_{0,i_1} \cdot Prob_{i_1,i_2}^{(\infty)} \cdot \ldots Prob_{i_{p-1},i_p}^{(\infty)}, \tag{3.2}$$

with $i_p = i_m$.

Proof The first statement follows from the remark that running the algorithm for n iterations, the probability of getting l_j is given by

$$(p_{i,j} + d_i(p_{0,j} - p_{i,j})) \cdot \sum_{l=0}^{n-1} t_i^l; \tag{3.3}$$

hence, as $n \to \infty$, we get

$$\frac{p_{i,j} + d_i(p_{0,j} - p_{i,j})}{1 - t_i}, \tag{3.4}$$

that is (3.1).

As to (3.2), note first that any trajectory with $i_p \neq i_m$ has probability zero of being constructed. In the case $i_p = i_m$, the probability $Prob_{tr}^{(n)}$ of the trajectory t by carrying out only n iterations is given in Lemma 3.1. Assuming by simplicity that $n - p$ is a multiple of p, we have

$$\sum_{j_1=0}^{(n-p)/p} t_{i_1}^{j_1} \cdot \ldots \cdot \sum_{j_{p-1}=0}^{(n-p)/p} t_{i_{p-1}}^{j_{p-1}} \leq \sum_{\substack{j_1,\ldots,j_{p-1}=0 \\ j_1+\ldots+j_{p-1}\leq n-p}}^{n-p} t_{i_1}^{j_1} \cdot \ldots \cdot t_{i_{p-1}}^{j_{p-1}} \leq \sum_{j_1=0}^{(n-p)} t_{i_1}^{j_1} \cdot \ldots \cdot \sum_{j_{p-1}=0}^{(n-p)} t_{i_{p-1}}^{j_{p-1}}. \tag{3.5}$$

As $n \to \infty$ we get

$$\sum_{j_1=0}^{\infty} t_{i_1}^{j_1} = \frac{1}{1 - t_{i_1}}, \ldots, \sum_{j_{p-1}=0}^{\infty} t_{i_{p-1}}^{j_{p-1}} = \frac{1}{1 - t_{i_{p-1}}}, \tag{3.6}$$

that leads us to (3.2). □

Finally we have the following theorem

Theorem 3.1 *The average number of function evaluations so that algorithm Glob finds the global minimum point is given by*

$$evals_2(d_1, \ldots, d_{m-1}) = \sum_{tr(\cdot)\in T} Prob_{tr}^{(\infty)}(\cdot) \left(k + f_{i_1} \frac{1}{Prob_{i_1,i_2}}, \ldots + f_{i_{p-1}} \frac{1}{Prob_{i_{p-1},i_p}} \right) \tag{3.7}$$

where

$$f_i = k \sum_{l=i+1}^{m} p_{i,l} + k d_i \left(1 - \sum_{l=i+1}^{m} p_{i,l} \right) + (1 - d_i) \left(1 - \sum_{l=i+1}^{m} p_{i,l} \right). \tag{3.8}$$

and T denotes the set of all feasible trajectories $tr(i_1, \ldots, i_p)$ whose last local minimum point is a global one.

Proof We can write

$$evals_2(d_1, \ldots, d_{m-1}) = \sum_{tr\in T} Prob_{tr}^{(\infty)}(\cdot)(f_0 + f_{i_1,i_2}, \ldots + f_{i_{p-1},i_p}), \tag{3.9}$$

where f_0 and $f_{i,j}$ denote the number of function evaluations needed to get the first local minimum and the average number of function evaluations to move from l_i to l_j, respectively. Further, $Prob_{tr}^{(\infty)}$ denotes the probability of trajectory tr to take place.

The number of function evaluations needed to get the first local minimum is

$$f_0 = k. \tag{3.10}$$

The average number of function evaluations $f_{i,j}$ to move from l_i to l_j is

$$f_{i,j} = f_i \frac{1}{Prob_{i,j}}, \quad i = 1, ..., m-1, \quad j = i+1, ..., m. \tag{3.11}$$

where f_i denotes the average number of function evaluations needed for not moving from l_i in a single choice of x_0 (that is for any iteration of *Glob*) and is given by

$$f_i = k \sum_{l=i+1}^{m} p_{i,l} + kd_i \left(1 - \sum_{l=i+1}^{m} p_{i,l}\right) + (1 - d_i)\left(1 - \sum_{l=i+1}^{m} p_{i,l}\right). \tag{3.12}$$

Clearly (3.10), (3.11), and (3.12) prove the theorem.

□

Now consider the following problem

Problem 3.1 *Let us consider Problem* 1.1 *and let the values* k, $p_{0,j}$ *and* $p_{i,j}$ *be given. Find values* d_i^*, $i = 1, ..., m-1$, *such that*

$$evals_2\left(d_1^*, ..., d_{m-1}^*\right) = \min_{d_1,...,d_{m-1}} evals_2\left(d_1, ..., d_{m-1}\right). \tag{3.13}$$

with $d_i \in I \equiv \{x \in R^n \mid 0 \le x_j \le 1,\ j = 1..., n\}$, $i = 1, ..., m-1$.

We can prove

Lemma 3.3 *$evals_2(d_1, ..., d_{m-1})$ attains its minimum at a vertex of the simplex I.*

Proof We just need to show that the derivative sign of $evals_2(d_1, ..., d_{m-1})$ with respect to d_i, $i = 1, \ldots, m-1$ is constant in the interval $[0, 1]$. We write (3.7) by pointing out the variable d_i, that is

$$\begin{aligned} evals_2(d_1, ..., d_{m-1}) = {} & k \sum_{tr \in T} Prob_{tr}^{(\infty)} \\ & + \sum_{tr \in T_{i,i+1}} Prob_{tr}^{(\infty)} \left(\frac{f_i}{Prob_{i,i+1}} + h_{tr}(\cdot)\right) \\ & \cdots \\ & + \sum_{tr \in T_{i,m}} Prob_{tr}^{(\infty)} \left(\frac{f_i}{Prob_{i,m}} + h_{tr}(\cdot)\right) \\ & + \sum_{tr \in T - T_i} g_{tr}(\cdot) \end{aligned} \tag{3.14}$$

where $h_{tr}(\cdot)$ denotes the sum of terms in $(k + f_{i_1} \frac{1}{Prob_{i_1,i_2}}, ...)$ of (3.7) that does not depend on d_i and $g_{tr}(\cdot)$ any term of the sum in (3.7). Further, T_i, $i = 1, \ldots, m-1$ is the set of the all trajectories that pass through the local minimum l_i and $T_{i,j}$, $j = i+1, \ldots, m$, denotes the set of all trajectories that move from l_i to l_j.

By the definition of $Prob_{tr}^{(\infty)}$, $Prob_{i,j}$ and f_i in (3.1) and (3.8), we can write for $j = i+1, \cdots, m$

$$\sum_{tr \in T_{i,j}} Prob_{tr}^{(\infty)} \left(\frac{f_i}{Prob_{i,j}} + h_{tr}(\cdot) \right)$$

$$= \frac{(k-1)\sum_{l=i+1}^{m} p_{i,l} + 1 + d_i \left(1 - \sum_{l=i+1}^{m} p_{i,l}\right)(k-1)}{\sum_{l=i+1}^{m} p_{i,l} + d_i \sum_{l=i+1}^{m} (p_{0,l} - p_{i,l})} \sum_{tr \in T_{i,j}} k_{tr}(\cdot)$$

$$+ \frac{p_{i,j} + d_i(p_{0,j} - p_{i,j})}{\sum_{l=i+1}^{m} p_{i,l} + d_i \sum_{l=i+1}^{m} (p_{0,l} - p_{i,l})} \sum_{tr \in T_{i,j}} k_{tr}(\cdot) h_{tr}(\cdot) \tag{3.15}$$

where $k_{tr}(\cdot)$ denotes the product of those terms in $Prob_{tr}^{(\infty)}$ that do not depend on d_i. Combining (3.14) and (3.15), since $\sum_{tr \in T} Prob_{tr}^{(\infty)} = 1$, we get

$$\begin{aligned} &evals_2(d_1, ..., d_{m-1}) \\ &= k + \frac{(k-1)\sum_{l=i+1}^{m} p_{i,l} + 1 + d_i \left(1 - \sum_{l=i+1}^{m} p_{i,l}\right)(k-1)}{\sum_{l=i+1}^{m} p_{i,l} + d_i \sum_{l=i+1}^{m} (p_{0,l} - p_{i,l})} \sum_{tr \in T_i} k_{tr}(\cdot) \\ &+ \frac{\sum_{j=i+1}^{m} \left(p_{i,j} \sum_{tr \in T_{i,j}} k_{tr}(\cdot) h_{tr}(\cdot)\right) + d_i \sum_{j=i+1}^{m} \left((p_{0,j} - p_{i,j}) \sum_{tr \in T_{i,j}} k_{tr}(\cdot) h_{tr}(\cdot)\right)}{\sum_{l=i+1}^{m} p_{i,l} + d_i \sum_{l=i+1}^{m} (p_{0,l} - p_{i,l})} \\ &+ \sum_{tr \in T - T_i} g_{tr}(\cdot). \end{aligned} \tag{3.16}$$

The latter may can be written in a compact way

$$evals_2(d_1, ..., d_{m-1}) = \theta + \frac{\alpha + \beta d_i}{\gamma + \delta d_i} \tag{3.17}$$

where α, β, γ, δ and θ are values that do not depend on d_i. The derivative of $evals_2(d_1, ..., d_{m-1})$ with respect to d_i is

$$evals_2(d_1, ..., d_{m-1})_{d_i} = \frac{\beta\gamma + \alpha\delta}{(\gamma + \delta d_i)^2}. \tag{3.18}$$

The lemma is clearly proven. □

We note that $evals_1(d_i)$ as given in (2.1) is part of the expression of $evals_2(d_1, ..., d_{m-1})$ in (3.16). We now report a counterexample that shows that an optimal value d_i for Problem 2.1 is not necessarily an optimal value for Problem 3.1.

Table 1 Probabilities values

$p_{0,1} = 0.5$	$p_{0,2} = 0.4$	$p_{0,3} = 0.1$
$p_{1,2} = 0.05$	$p_{1,3} = 0.04$	$p_{2,3} = 0.01$

Counterexample 3.1 Consider the case $m = 3$. In this case we can have four trajectories and the set T is given by

$$T = \{(3), (1, 2, 3), (1, 3), (2, 3)\}. \tag{3.19}$$

From equation (3.16) we find

$$\begin{aligned} evals_2(d_1, d_2) = & \\ & + k + p_{0,1}\frac{(k-1)(p_{1,2}+p_{1,3})+1+d_1(k-1)(1-p_{1,2}-p_{1,3})}{p_{1,2}+p_{1,3}+d_1(p_{0,2}-p_{1,2}+p_{0,3}-p_{1,3})} \\ & + p_{0,1}\frac{p_{1,2}+d_1(p_{0,2}-p_{1,2})}{p_{1,2}+p_{1,3}+d_1(p_{0,2}-p_{1,2}+p_{0,3}-p_{1,3})} \\ & \cdot\frac{(k-1)p_{2,3}+1+d_2(k-1)(1-p_{2,3})}{p_{2,3}+d_2(p_{0,3}-p_{2,3})} \\ & + p_{0,2}\left(\frac{(k-1)p_{2,3}+1+d_2(k-1)(1-p_{2,3})}{p_{2,3}+d_2(p_{0,3}-p_{2,3})}\right). \end{aligned} \tag{3.20}$$

By (2.1) we get

$$evals_1\_d_1(d_1) = \frac{(k-1)(p_{1,2}+p_{1,3})+1+d_1(k-1)(1-p_{1,2}-p_{1,3})}{p_{1,2}+p_{1,3}+d_1(p_{0,2}-p_{1,2}+p_{0,3}-p_{1,3})}, \tag{3.21}$$

$$evals_1\_d_2(d_2) = \frac{(k-1)p_{2,3}+1+d_2(k-1)(1-p_{2,3})}{p_{2,3}+d_2(p_{0,3}-p_{2,3})}. \tag{3.22}$$

By choosing values $p_{i,j}$ as given in Table 1, and evaluating $evals_2(d_1, d_2)$ at the vertices of the simplex I we get Table 2. Further by evaluating $evals_1\_d_1(d_1)$ and $evals_1\_d_2(d_2)$ at the endpoints of [0.1] we get Table 3.

From Tables 2 and 3 we can see that for $k = 16$ and $k = 4$, $evals_2(d_1, d_2)$, $evals_1\_d_1(d_1)$ and $evals_1\_d_2(d_2)$ attain the minimum at the same values of d_1 and d_2. That does not hold for $k = 8$.

Table 2 Evaluations of $evals_2(\cdot)$ at the vertices

k	$evals_2(0, 0)$	$evals_2(0, 1)$	$evals_2(1, 0)$	$evals_2(1, 1)$
16	120.06	150.56	140.00	176.00
8	98.63	80.33	109.60	88.00
4	87.92	45.22	87.92	44.00

Table 3 Evaluations of $evals_1\_d_1(\cdot)$ and $evals_1\_d_2(\cdot)$ at 0 and 1

k	$evals_1\_d_1(0)$	$evals_1\_d_1(1)$	$evals_1\_d_2(0)$	$evals_1\_d_2(1)$
16	26.11	32.00	115.00	160.00
8	18.11	16.00	107.00	80.00
4	14.11	8.00	103.00	40.00

4 Numerical results

In this section a parallel version of Algorithm 2.1, was presented. The algorithm follows the multiple instructions, multiple data (MIMD) model; in an environment of N processors there is one server and N-1 clients. The server accomplishes the following

- reads all the initial data and sends them to each client;
- receives the intermediate data from a sender client;
- combines them with all the data already received;
- sends back the updated data to the client sender;
- gathers the final data from each client.

while each client executes the following

- receives initial data from server;
- runs algorithm *Glob*;
- sends intermediate data to server;
- receives updated values from server;
- stops running *Glob* whenever its stop rule is met in any client execution;
- sends final data to server.

The communication that takes place between the server and each client concerns mainly the parameters in (2.4), that is, p_2, p_3 and k. Each client, as soon as it either finds a new local minimizer or a fixed number of iterations have been executed, sends a message to the server containing the data gathered after the last sent message; that is

- last minimum found;
- the number of function evaluations since last message sending;
- the number of iterations since last message sending;
- the number of local searches carried out since last message sending;
- status variable of value 0 or 1 denoting that the stop rule has been met.

The server combines each set of intermediate data received with the ones stored in its memory and sends to the client the new data. If the server receives as status variable 1 in the subsequent messages sent to clients the status variable will keep the same value, meaning that the client has to stop running *Glob* and has to send the final data to the server.

The parallel version of Algorithm 2.1 has been tested in a parallel MatLab environment under the Linux operating system.

The following test problems have been considered.

Problem 4.1

$$minf(x) = \frac{\pi}{n}\left\{10sin^2(\pi y_1) + \sum_{i=1}^{n-1}\left[(y_i-1)^2\left(1+10sin^2(\pi y_{i+1})\right)\right] + (y_n-1)^2\right\}$$

with $n = 100$, $y_i = 1 + \frac{1}{4}(x_i - 1)$, $S \equiv \{x \in R^n \mid -10 \le x_i \le 10, \ \ i = 1, ..., n\}$;

Problem 4.2

$$minf(x) = (x_1 - 1)^2 + \sum_{i=2}^{n} i\left(2x_i^2 - x_{i-1}\right)^2$$

with $n = 25$, $S \equiv \{x \in R^n \mid -10 \le x_i \le 10, \ \ i = 1, ..., n\}$;

Problem 4.3

$$minf(x) = 10n + \sum_{i=1}^{n}\left(x_j^2 - 10cos\left(2\pi x_j\right)\right);$$

with $n = 8$, $S \equiv \{x \in R^n \mid -2.56 \le x_i \le 2.56, \ \ i = 1, ..., n\}$;

Problem 4.4

$$minf(x) = \begin{cases} \left(\dfrac{2}{\rho_i^2}\dfrac{\langle x - m_i, x_t - m_i\rangle}{\|x - m_i\|} - \dfrac{2}{\rho_i^3}(\|x_t - m_i\|^2 + 4 - f_i)\right) & \\ \quad \times\|x - m_i\|^3 + & \\ \left(1 - \dfrac{4}{\rho_i}\dfrac{\langle x - m_i, x_t - m_i\rangle}{\|x - m_i\|} + \dfrac{3}{\rho_i^2}(\|x_t - m_i\|^2 + 4 - f_i)\right) & \\ \quad \times\|x - m_i\|^2 + f_i, & x \in B_i, \forall i \\ \|x - x_t\|^2 + 4, & x \notin B_i, \forall i, \end{cases}$$

with $n = 20$, $B_i \equiv \{x \in \mathbb{R}^n \mid \|x - m_i\| \le \rho_i\}$, *for* $i = 1, ..., 9$, $S \equiv \{x \in R^n \mid -1 \le x_j \le 1, \ j = 1, \ldots, n\}$, m_i, $(i = 1, ..., 9)$, *and* x_t *denoting ten points uniformly chosen in* S *such that the* B_i *balls do not overlap each other,* f_i *real values to be taken as the values of* $f(\cdot)$ *at* m_i.

Problems 4.1, 4.2, and 4.3 appeared in [5, 12] and [16] respectively. Problem 4.4 belongs to a family of problems introduced in [7] and implemented in the software GKLS (cfr. [6]).

The local minimization has been carried out by a code, called *cgtrust*, written by C.T. Kelley [11]. This code implements a *trust region* type algorithm that uses a polynomial procedure to compute the step size along a search direction. The *cgtrust* code was written by Kelley according to the MatLab programming

language. In order to speed up execution, this has been converted in the C language and by using the *mex* MatLab utility, compiled so that it could be run as a built-in function. Two computers have been used; the first equipped with an Intel Quad CPU Q9400 based on four processors, the second with a AMD PHENOM II X6 1090T based on six processors. Experiments have been carried out both on each single computer and on the two connected to a local network. In Table 4 we report the fourteen configurations of the computing resources used in each of our experiments. We shall use the MatLab notation that denotes a processor as a *worker* available in the configuration.

The code used when using just one worker does not make any reference to Matlab parallel tools; as a consequence, the code is much simpler than the one implemented for more than one worker. Since our algorithm makes use of random procedures, to get significant results in solving the test problems, 100 runs of the algorithm have been done on each problem. The data reported in the tables are all mean values. The parameter k that evaluates the computational cost of local searches has been computed as the sum of function and gradient evaluations of the current objective function. The algorithm stops whenever the global minimum has been found within a fixed accuracy. That is the stop rule is

$$|f^* - \bar{f}| < \epsilon_1 |f^*| + \epsilon_2, \quad \epsilon_1 = 10^{-3}, \ \epsilon_2 = 10^{-5}$$

with f^* and $\bar{f}$ function values at the global minimum point and at the last local minimum found.

To evaluate the performance of our algorithm we consider two indices, the speedup and the efficiency; the first estimates the decrease of the time of a parallel execution with respect to a sequential run. The second index estimates how much the parallel execution exploit the computer resources. In Tables 5, 6, 7, and 8 we report the results gathered for each configuration given in

Table 4 The configurations of the computing resources

Config. no.	Computer 1	Computer 2	Workers in 1	Workers in 2
1	Intel Quad		1	
2	Intel Quad		2	
3	Intel Quad		3	
4	Intel Quad		4	
5	Amd phenom 6		1	
6	Amd phenom 6		2	
7	Amd phenom 6		4	
8	Amd phenom 6		6	
9	Intel Quad	Amd phenom 6	2	2
10	Intel Quad	Amd phenom 6	4	4
11	Intel Quad	Amd phenom 6	4	6
12	Amd phenom 6	Intel Quad	2	2
13	Amd phenom 6	Intel Quad	4	4
14	Amd phenom 6	Intel Quad	6	4

Table 5 Results working with Amd Phenom 6

procs	fun_1			fun_2			fun_3			fun_4		
	secs	*speed*	*eff*	*secs*	*speed*	*eff*	*secs*	*speed*	*eff*	*secs*	*speed*	*eff*
1	0.06			472.2			478.1			84.50		
2	0.08	0.70	0.35	618.6	0.76	0.38	685.5	0.70	0.35	90.80	0.93	0.47
4	0.03	2.28	0.57	196.7	2.40	0.60	225.5	2.12	0.53	34.78	2.43	0.61
6	0.02	2.59	0.43	122.9	3.84	0.64	146.7	3.26	0.54	19.87	4.25	0.71

Table 6 Results working with Intel Quad

procs	fun_1			fun_2			fun_3			fun_4		
	secs	*speed*	*eff*	*secs*	*speed*	*eff*	*secs*	*speed*	*eff*	*secs*	*speed*	*eff*
1	0.15			803.6			820.9			143.0		
2	0.12	1.28	0.64	996.9	0.81	0.40	855.5	0.96	0.48	154.8	0.92	0.46
3	0.03	5.57	1.86	520.5	1.54	0.51	482.2	1.70	0.57	84.20	1.70	0.57
4	0.03	5.74	1.44	280.6	2.86	0.72	338.0	2.43	0.61	59.76	2.39	0.60

Table 7 Results working with Intel Quad and Amd Phenom 6

procs	fun_1			fun_2			fun_3			fun_4		
	secs	*speed*	*eff*	*secs*	*speed*	*eff*	*secs*	*speed*	*eff*	*secs*	*speed*	*eff*
1	0.10			637.9			649.5			113.8		
$2+2$	0.03	3.95	0.99	307.2	2.08	0.52	275.6	2.36	0.59	54.68	2.08	0.52
$4+4$	0.03	3.85	0.48	131.9	4.84	0.60	134.6	4.83	0.60	24.23	4.70	0.59
$4+6$	0.03	4.11	0.41	83.56	7.63	0.76	96.12	6.76	0.68	18.92	6.01	0.60

Table 8 Results working with Amd Phenom 6 and Intel Quad

procs	fun_1			fun_2			fun_3			fun_4		
	secs	*speed*	*eff*	*secs*	*speed*	*eff*	*secs*	*speed*	*eff*	*secs*	*speed*	*eff*
1	0.10			637.9			649.5			113.8		
$2+2$	0.03	3.92	0.98	255.9	2.49	0.62	287.7	2.26	0.56	48.1	2.37	0.59
$4+4$	0.03	3.95	0.49	133.7	4.77	0.60	116.6	5.57	0.70	19.18	5.93	0.74
$6+4v$	0.026	4.06	0.41	90.50	7.05	0.70	109.6	5.93	0.59	13.69	8.31	0.83

Table 4; in each table for each function we report the computational expired time, the speedup and the efficiency. Note that the rows in Tables 7 and 8 referring to one worker have been calculated as the means of the values in the corresponding rows in Tables 5 and 6.

From the data in the tables we can make the following remarks.

- In almost all the experiments the parallel algorithm improves largely the speedup of the computation. The efficiency in many experiments is above 0.70 although the task of a worker is to start the process and to collect and to distribute the intermediate and final data.
- The speedup becomes less than one only when two workers are employed. Clearly this has to be related to the fact that the complexity of the parallel code is not balanced by the use of just one additional worker.

References

1. Boender, C.G.E., Rinnooy Kan, A.H.G.: Bayesian stopping rules for a class of stochastic global optimization methods. Technical Report Report 8319/0, Erasmus University Rotterdam (1985)
2. Cetin, B.C., Barhen, J., Burdick, J.W.: Terminal repeller unconstrained subenergy tunnelling (trust) for fast global optimization. J. Optim. Theory Appl. **77**(1), 97–126 (1993)
3. Desai, R., Patil, R.: Salo: combining simulated annealing and local optimization for efficient global optimization. In: Proceedings of the 9th Florida AI Research Simposium, FLAIRS-96', pp. 233–237 (1996)
4. Eskow, E., Schnabel, R.B.: Mathematical modelling of a parallel global optimization algorithm. Parallel Comput. **12**(3), 315–325 (1989)
5. Floudas, C.A., Pardalos, P.M.: A Collection of Test Problems for Constraint Global Optimization Algorithms. Springer, Berlin Heidelberg (1990)
6. Gaviano, M., Kvasov, D.E., Lera, D., Sergeyev, Y.D.: Software for generation of classes of test functions with known local and global minima for global optimization. ACM Trans. Math. Softw. **29**(4), 469–480 (2003)
7. Gaviano, M., Lera, D.: Test functions with variable attraction regions for global optimization problems. J. Glob. Optim. **13**(2), 207–223 (1998)
8. Gaviano, M., Lera, D., Steri A.M.: A local search method for continuous global optimization. J. Glob. Optim. **48**, 73–85 (2010)
9. Gergel, V.P., Sergeyev, Ya.D.: Sequential and parallel global optimization algorithms using derivatives. Comput. Math. Appl. **37**(4/5), 163–180 (1999)
10. Hedar, A.R., Fukushima, M.: Tabu search directed by direct search methods for non linear global optimization. Eur. J. Oper. Res. **170**(2), 329–349 (2006)
11. Kelley, C.T.: Iterative Methods for Optimization. SIAM, Philadelphia (1999)
12. Levy, A., Montalvo, A.: The tunneling algorithm for the global minimization of functions. SIAM J. Sci. Statist. Comput. **6**, 15–29 (1985)
13. Lucidi, S., Piccioni, M.: Random tunneling by means of acceptance-rejection sampling for global optimization. J. Optim. Theory Appl. **62**(2), 255–277 (1989)
14. Nemirovsky, A.S., Yudin, D.B.: Problem Complexity and Method Efficiency in Optimization. John Wiley and Sons, Chichester (1983)
15. Oblow, E.M., Spt: a stochastic tunneling algorithm for global optimization. J. Glob. Optim. **20**, 195–212 (2001)
16. Rastrigin, L.A.: Systems of Extreme Control. Nauka, Moscow (1974)
17. Rudolph, G.: Parallel approaches to stochastic global optimization. In: Joosen, W., Milgrom, E. (eds.) Parallel Computing: from Theory to Sound Practice, pp. 256–267. IOS (1992)

18. Sergeyev, Ya.D.: Parallel information algorithm with local tuning for solving multidimensional go problems. J. Glob. Optim. **15**(2), 157–167 (1999)
19. Vavasis, S.A.: Complexity issues in global optimization: a survey. In: Horst, R., Pardalos, P.M. (eds.) Handbook of Global Optimization, pp. 27–41. Kluwer Academic Publishers (1995)
20. Čiegis, R., Baravykaité, M., Belevičius, R.: Parallel global optimization of foundation schemes in civil engineering. In: Applied Parallel Computing. State of the Art in Scientific Computing, Lecture Notes in Computer Science, vol. 3732/2006, pp. 305–312 (2006)