J Glob Optim (2009) 44:461–479
DOI 10.1007/s10898-008-9329-3

A hybrid multiagent approach for global trajectory optimization

Massimiliano Vasile · Marco Locatelli

Received: 25 May 2007 / Accepted: 12 March 2008 / Published online: 8 August 2008
© Springer Science+Business Media, LLC. 2008

Abstract In this paper we consider a global optimization method for space trajectory design problems. The method, which actually aims at finding not only the global minimizer but a whole set of low-lying local minimizers (corresponding to a set of different design options), is based on a domain decomposition technique where each subdomain is evaluated through a procedure based on the evolution of a population of agents. The method is applied to two space trajectory design problems and compared with existing deterministic and stochastic global optimization methods.

Keywords Global optimization · Space trajectory design · Agent-based approach · Domain decomposition

1 Introduction

Methods and tools for preliminary trajectory design have recently become an important topic within the space community. Space mission design is subdivided into different phases. The first one is a mission feasibility study, which has to analyze, in a reasonably short time, a large number of different mission options. Each mission option requires the design of one or more optimal trajectories. All this can be reformulated as a global optimization problem or, more precisely, a global search for multiple low-lying local minimizers. Space trajectory design problems have been tackled with some global optimization methods for

M. Vasile (✉)
Department of Aerospace Engineering, University of Glasgow, James Watt South Building, Glasgow G12 8QQ, UK
e-mail: m.vasile@eng.gla.ac.uk

M. Locatelli
Dipartimento di Informatica, Università di Torino, Corso Svizzera 185, Torino 10149, Italy
e-mail: locatelli@di.unito.it

more than 10 years already but a systematic comparison of different approaches was performed only recently [6,15]. Often genetic algorithms have been the preferred option. In particular, Gage et al. have shown the effectiveness of genetic algorithms with niching technique compared to a simple grid search for the optimization of bi-impulsive transfers [8], Coverstone et al. used genetic algorithms for low-thrust trajectory design [10,18], Gurfil et al. used niching genetic algorithms for the characterization of geocentric orbits [9], Vasile proposed a hybridization of Evolutionary Algorithms with SQP methods for the design of low energy transfers to the Moon [23], Rogata et al. proposed an implementation of GAs for the design of multiple gravity assist trajectories [19], and Dachwald combined neurocontrollers with Evolutionary Algorithms for the design of low-thrust trajectories [2]. However, it has been recently shown that Differential Evolution outperforms GAs on some trajectory design problems [6,15]. Moreover, it was demonstrated that a hybridization between these global optimization techniques, generally applicable to black-box problems, with ad hoc branch-and-prune methods and exploiting the properties of the problem (e.g., continuity and differentiability, periodicity, symmetry, modularity) can greatly improve convergence [15].

Evolutionary-based approaches usually mimic behaviors observed in nature. From the very basic evolutionary paradigms to the more complex behaviors of ant colonies or bird flocks, each one of these heuristics can be interpreted as basic behaviors (like reproduction, feeding or trail following) associated to individual agents. This paper presents a generalization of this concept: a population of agents is endowed with a set of individualistic and social behaviors, in order to explore a virtual environment made up of the solution space. This particular meta-heuristic is inspired by the control of multiple robotic agents. The combination of individualistic and social behaviors aims at balancing local and global search (or exploitation and exploration), which is the key issue for any heuristic for global optimization problems.

We remark that during the preliminary design phase there is a need for multiple mission options. Therefore, as already pointed out before, instead of looking for the global minimizer, the proposed approach will store a set of low-lying local minimizers, which (hopefully) contains the global one. The proposed meta-heuristic was also hybridized with a domain decomposition technique in order to increase the exploration phase.

The paper is organized as follows. In Sect. 2 we give a short description of two space trajectory design problems which will be used as test problems for the proposed method. In Sect. 3 we describe the domain decomposition technique. In Sect. 4 we describe the MultiAgent Collaborative Search (MACS) which is employed to evaluate each subdomain and is based on the evolution of a population of agents. In Sect. 5 the proposed approach is compared with some well-known stochastic and deterministic methods, paying particular attention to optimally tuning the typical parameters of each method (e.g., population size, mutation probability in GAs).

2 Description of two space trajectory design problems

In this section we consider two different mathematical models for the design of an interplanetary transfers trajectory. In both these models the objective will be to minimize the variation of the velocity of the spacecraft due to a propelled manoeuvre, or Δv in the following. Minimizing the Δv means minimizing the propellant mass required to perform the manoeuvre, since propellant mass increases exponentially with Δv.

2.1 Bi-impulsive orbital transfers

A simple, but already significant, application is to find the best launch date and time of flight to transfer a spacecraft from one celestial body (say the Earth) to another one (say Mars or an asteroid). This can be briefly sketched as follows: A spacecraft initially moves around the Sun along the orbit of the departure celestial body. It is assumed that the celestial body is just a point in space with no mass, therefore the position of the spacecraft coincides with that of the celestial body. The spacecraft is injected into an elliptical orbit through a change in its heliocentric velocity. This elliptical orbit, or transfer orbit, is designed to intersect the orbit of the destination celestial body. At the destination a second change in velocity would inject the spacecraft into the orbit of the target celestial body. Even the arrival celestial body, as the departure one, is assumed to be a point in space with no mass. Each change in the heliocentric velocity is called a Δv.

In this problem we only have two *control variables*.

- t_0, the departure date of the spacecraft;
- T_1, the flight time from the first celestial body to the second one;

Given t_0, the departure date from the first celestial body, and $t_0 + T_1$, the arrival date at the second one, the position and velocity of the two bodies at t_0 and $t_0 + T_1$ respectively, can be computed through analytical ephemeris (i.e., analytical formulas giving the position and the velocity vectors of a celestial body as a function of time). Then, given the flight time T_1 from the first to the second body, the transfer orbit can be computed solving a Lambert's problem [1]. We are interested only in direct flights with zero revolutions around the Sun, therefore, apart from the special cases in which the transfer is at 180 or 360°, the solution of the Lambert's problem is unique. The solution of the Lambert's problem provides the heliocentric velocities at the beginning $\mathbf{v}_1$ and at the end $\mathbf{v}_2$ of the transfer arc. Since the spacecraft initially moves along the orbit of the first body with the same velocity $\mathbf{v}_{p1}$, the initial Δv is

$$\boldsymbol{\Delta v}_1 = \mathbf{v}_1 - \mathbf{v}_{p1},$$

and the second one in order to move from the transfer orbit to the final one is

$$\boldsymbol{\Delta v}_2 = \mathbf{v}_2 - \mathbf{v}_{p2},$$

Their norms $\Delta v_1 = \|\mathbf{v}_1 - \mathbf{v}_{p1}\|$ and $\Delta v_2 = \|\mathbf{v}_2 - \mathbf{v}_{p2}\|$ are the two contributions to the total $\Delta v = \Delta v_1 + \Delta v_2$. In general in order to transfer a spacecraft from celestial body one to celestial body two, the sum of both Δv's should be minimized. However in the following we analyze the case of a spacecraft ramming into an asteroid, therefore the *objective function* for this first problem is simply the first change in velocity:

$$\Delta v_1 \tag{1}$$

The reason for this choice is that for the proposed test case, the difficulty of the search for a minimum increases. Our problem will be to minimize (1) subject to simple bounds over t_0 and T_1.

2.2 Multi gravity assist trajectories

A common problem in space mission analysis is the optimal design of transfer trajectories exploiting one or more *gravity assist manoeuvres* (MGA) to change the orbital parameters of a spacecraft. Each gravity manoeuvre occurs at a planet and exploits the gravity action of the planet to produce a Δv. Since Δv's are normally produced by propelled manoeuvres, optimal sequences of gravity assist manoeuvres minimize the total propelled Δv required to reach a given target. A general gravity assist manoeuvre in the solar system can be modeled assuming the planet to be a point mass and the manoeuvre to be instantaneous. In this simplified model, the gravity action produces a simple rotation of the incoming velocity vector, relative to the planet, in the plane identified by the incoming and the outgoing velocity vectors. This rotation is normally a function of the modulus of the incoming velocity and of the minimum distance of the spacecraft relative to the planet [14]. Due to this functional dependency, caused by physical reasons, not all the desired rotations can be achieved. Therefore, an additional propelled manoeuvre is generally necessary to correct the outgoing velocity vector. The total cost of a MGA transfer can be defined as the sum of all the corrective Δv_i^c manoeuvres for all the N_p planetary encounters plus the initial Δv_0 at launch from Earth,

$$f = \Delta v_0 + \sum_{i=1}^{N_p} \Delta v_i^c \tag{2}$$

The conic arc connecting two subsequent planets i and $i+1$ at positions $\mathbf{R}_i$ and $\mathbf{R}_{i+1}$ is computed as a Lambert's [1] solution. The two position vectors are functions of the encounter dates t_i and t_{i+1}. Therefore, the solution vector x is defined by the launch date t_0 and by all the times of flight (TOF) from one planet to the subsequent one $T_i = t_{i+1} - t_i$:

$$x = [t_0, T_1, \ldots, T_{N_p-1}] \tag{3}$$

In the computational experiments presented in Sect. 5 the sequence of planets is equal to that of the Cassini mission to Saturn, i.e., Earth-Venus-Venus-Earth-Jupiter-Saturn, and is fixed. We also point out that the constraints on the minimum distance from the center of each planet, i.e., the pericentre radius γ_i, were directly included in the objective function through appropriate penalty terms. Therefore, objective function (2) was modified as follows (see [15])

$$\begin{aligned} f = \; & \Delta v_0 + \sum_{i=1}^{4} \Delta v_i^c + w_1 \max\{0, 6351.8 - \gamma_1\} + w_2 \max\{0, 6351.8 - \gamma_2\} \\ & + w_3 \max\{0, 6778.1 - \gamma_3\} + w_4 \max\{0, 671492.0 - \gamma_4\}, \end{aligned} \tag{4}$$

where $w_1, \ldots, w_4$ are given penalty parameter values fixed in [15] as follows

$$w_1 = w_2 = w_3 = 0.01 \quad w_4 = 0.001.$$

Although in the computational experiments we will concentrate on the problem described above, we point out that such problems can be generalized in a number of ways. For instance, *deep space maneuvers*, i.e., variations of the velocity which can be introduced when the spacecraft is in the deep space (far from the planets) can be added. The starting time and the time intervals between subsequent velocity variations will always be variables of the problems. In more complicated models the sequence of planets is not fixed in advance so that we need to introduce further (discrete) variables corresponding to the choice of this sequence.

As already remarked in the Introduction, we underline again that, whatever problem we try to solve, we are usually interested not only in the global minimizer but also in the identification of low-lying local minimizers which offer a set of possible choices.

3 A solution method combining branching and an agent-based search

The problems will be tackled with a combination of two different techniques: a deterministic and a stochastic one. The deterministic method proposed in this paper is based on branching, i.e., subdivision of the feasible region into smaller and smaller subdomains. Since the test problems considered in this paper can be reformulated as box-constrained ones, the feasible region of each problem, denoted by D, is a hyperrectangle and the subdomains into which it is subdivided are also hyperrectangles. The stochastic algorithm is a multiagent-based one and searches on the subdomains in order to evaluate them. In this section we will give the details of the deterministic method, while the multiagent approach will be described in the following section.

3.1 The branching procedure

Below we give the description of a generic branching procedure for GO problems.
BRANCHING PROCEDURE

Step 0. Initialization: Let $\mathcal{F} = \{D\}$.
Step 1. Node selection: Let θ be a function which associates a value to each node $H \in \mathcal{F}$. Then, select a node $\overline{H} \in \mathcal{F}$ such that

$$\overline{H} \in \arg\min_{H \in \mathcal{F}} \theta(H), \tag{5}$$

Step 2. Evaluation: Evaluate the selected node $\overline{H}$ through some procedure.
Step 3. Node branching: Subdivide $\overline{H}$ into η nodes $H_i,\ i = 1, \ldots, \eta$, for some integer $\eta \geq 2$, and update $\mathcal{F}$ as follows:

$$\mathcal{F} = (\mathcal{F} \setminus \{\overline{H}\}) \cup \{H_1, \ldots, H_\eta\}.$$

Step 4. Node deletion: Delete nodes from $\mathcal{F}$ according to some rule.
Step 5. Stopping rule: If $\mathcal{F} = \emptyset$, then STOP. Otherwise, go back to Step 1.

Note that in the scheme above each node corresponds to a subdomain and in what follows the two terms will be used as synonymous. Such scheme is quite typical for branch-and-bound methods. For these methods θ delivers a lower bound for each subdomain; each node is evaluated by evaluating feasible points within the corresponding subdomain (if any) and possibly updating the upper bound; node branching can be performed in several ways; node deletion is done through standard fathoming rules. However, what is missing in our context is an easy way to obtain bounds. Therefore, while we retain the branching structure, we need some other ways to define a function θ and to evaluate, branch and delete nodes. All this will be specified in the following subsections.

3.1.1 Node evaluation

The evaluation of a subdomain H is done by running a MultiAgent Collaborative Search (MACS) algorithm within H. As will be explained in details in Sect. 4, the MACS algorithm explores the subdomain H and stores in an archive X all the promising points in H. After N_f function evaluations, the final population P_{fin} of agents, used to explore H, is added to X and the points in $E_v(H) = X \cap H$ represent the evaluation of the subdomain.

3.1.2 Node branching

First we recall that each subdomain is a hyperrectangle. Branching is done through the standard bisection method: the (relative) largest edge of the domain to be subdivided is selected and two new subdomains (i.e, $\eta = 2$) are obtained by splitting with respect to its midpoint. More formally, let

$$H = [a_1, b_1] \times \cdots \times [a_n, b_n]$$

be the domain to be subdivided. Let

$$j \in \arg \max_{i=1,\ldots,n} \frac{b_i - a_i}{D_i - d_i},$$

where d_i and D_i denote respectively the lower and upper bounds for variable x_i in the original domain D. Let

$$\tilde{x}_j = \frac{a_j + b_j}{2}$$

be the midpoint of edge $[a_j, b_j]$. Then, we define the two new subdomains

$$H_1 = [a_1, b_1] \times \cdots \times [a_j, \tilde{x}_j] \times \cdots \times [a_n, b_n],$$

$$H_2 = [a_1, b_1] \times \cdots \times [\tilde{x}_j, b_j] \times \cdots \times [a_n, b_n].$$

3.1.3 The function θ

Before defining the function θ we need to introduce two other functions ω and φ. Let H be a given subdomain and $\tilde{H}$ be its father. Function ω is defined as follows for H:

$$\omega(H) = \frac{\max\{N(H), 1\}}{N} \frac{\ell(D)}{\ell(H)} \tag{6}$$

where $\ell(\cdot)$ denotes the geometric mean of the edge lengths of an n-dimensional hyperrectangle, N is the number of points in $E_v(\tilde{H})$, obtained through the evaluation of the father node $\tilde{H}$ by the MACS algorithm, and $N(H)$ is equal to the number of points in $E_v(\tilde{H})$ which also belong to H, i.e., $N(H) = \mid E_v(\tilde{H}) \cap H \mid$ (for the root node D we simply set $N(D) = N$). We also remark that in (6) the ratio between geometric means of the edge lengths can also be viewed as the nth root of a ratio between volumes:

$$\frac{\ell(D)}{\ell(H)} = \left(\frac{Vol(D)}{Vol(H)}\right)^{\frac{1}{n}}.$$

Then, small values for ω are obtained for subdomains with small $N(H)$ and large volume, i.e., for subdomains with a low density of observed points. Function φ is defined as follows:

$$\varphi(H) = \begin{cases} \frac{f^H_{best} - f_{best}}{f_{worst} - f_{best}} & \text{if } N(H) > 0 \\ 1 & \text{otherwise} \end{cases} \tag{7}$$

where f_{best} and f_{worst} are respectively the best (lowest) and worst (highest) function values observed up to now by the algorithm within the feasible region, and f^H_{best} denotes the best observed function value in $H \cap E_v(\tilde{H})$ (if any). For $N(H) > 0$, φ returns a value between 0 and 1, while for $N(H) = 0$ the φ value is simply set to 1.

Now we are ready to define function θ. This is a convex combination of the two above functions, i.e.,

$$\theta(H) = \sigma\omega(H) + (1-\sigma)\varphi(H), \tag{8}$$

where $\sigma \in [0, 1]$ is a parameter which controls the relative weights of ω and φ. Note that the choice of σ allows us to balance global and local exploration of the algorithm. If σ is close to 0 in (8), φ has a higher weight and nodes where good function values have been observed are favoured (local exploration), while when σ is close to 1, ω has the higher weight in (8) and nodes with high volume and few observed points of their father node are favoured (global exploration, i.e., exploration of regions with a low density of observed points). The presence of a parameter balancing local and global exploration is quite typical for GO methods (see, e.g., [13]).

3.1.4 Node deletion

We employed a very simple rule to delete nodes: all nodes are deleted at the same time when the number of explored subdomains reaches a predefined number τ. This deletion rule is clearly of heuristic nature, although other rules are possible. For instance, we could fix a maximum level τ of the branching tree: a node H is deleted if it is the result of τ previous subdivisions. Or, alternatively, a node could be deleted if it is the result of τ consecutive subdivisions where no improvement has been observed. However, the current simple rule turned out to be already appropriate.

3.2 Convergence of the branching algorithm

Convergence of a GO method to a global minimiser is, admittedly, often not very interesting from the practical point of view, because such convergence can only be guaranteed after unacceptably long computation times. However, for the sake of completeness we prove in this section the convergence of the above described branching procedure, independently of the algorithm employed to evaluate the nodes of the tree, when we remove the node deletion step and, consequently, run the algorithm for an infinite amount of time. As remarked, e.g., in [20,22], in all cases where no global information about the objective function, such as the value of its Lipschitz constant, is available, the only way to ensure that a GO method is convergent is that the set of points at which the function is observed is dense within the feasible region. In the branching procedure described above this can be guaranteed if the branching mechanism is *exhaustive*, i.e., each nested sequence of subdomains in $\mathcal{F}$ converges to a single point if the algorithm is never stopped (see, e.g., [11]). In order to prove exhaustiveness, we need two lemmas. The branching process generates a tree. For a given subdomain H, the level in the tree of the node corresponding to H is equal to the number of subdivisions which lead from the original domain D to subdomain H. The first lemma states that, given an infinite nested sequence of subdomains $\{H_k\}$, where $H_0 = D$ and k denotes the level of the subdomain, as k increases to infinity, the sequence converges to a single point. The proof is omitted because is a trivial consequence of the simple rule (bisection of the longest edge) employed.

Lemma 1 *Let $\{H_k\}$ be an infinite nested sequence of subdomains. Then:*

$$\cap_{k=1}^{\infty} H_k = \{x\},$$

for some $x \in D$.

The next lemma proves that no finite nested sequence of nodes can be generated by the branching procedure.

Lemma 2 *If $\sigma > 0$ in (8), all the nested sequences generated by an infinite run of the algorithm are infinite ones.*

Proof The proof is by contradiction. Assume that a finite nested sequence exists. Let K be the last level of the sequence and H_K the last subdomain of the sequence. The value θ for H_K is bounded from above by a constant $\overline{\theta} > 0$. Since each time we move down one level an edge is halved, ω and, consequently, also the θ value increase to infinity as the levels increase to infinity. Therefore, there exists some level $K' > K$ such that each node at a level $k \geq K'$ has a value θ greater than $\overline{\theta}$. Then, according to the selection rule (5), only nodes at a level lower than K' can be selected. But these are in finite number and after a finite number of iterations we will have to select H_K to be subdivided, which is a contradiction.

Now the proof of convergence immediately follows from the above lemmas. Indeed, since all nested sequences are infinite ones, and all infinite nested sequences converge to a point in D, then exhaustiveness is proved and the following theorem immediately follows.

Theorem 1 *Assume $\sigma > 0$ in (8). Then, if the branching procedure is never stopped and no node is deleted, the best feasible point observed while running the procedure converges to the global minimiser of the problem.*

4 Multiagent collaborative search

Some stochastic approaches implement biologically or physically inspired heuristics to drive the search toward potentially interesting portions of the solution space (e.g., evolutionary algorithms, particle swarm optimization, ant colony optimization, simulated annealing). Other stochastic approaches consider a combination of random sampling with local optimization (variable neighborhood search, basin hopping, multistart). Each one of these methods has a set of operators that are devoted to the local exploration of the solution space (e.g., the local optimizer for VNS) and other operators that are in charge of the global exploration. In general each operator is conceived to allow an automatic decision on how to proceed with the search. Therefore, all the above mentioned stochastic methods can be generally considered to implement a decision-making process which yields sets of actions and exploits the information that those actions give in return.

In this paper the general framework described above will be implemented through a multiagent approach. In order to explore each subdomain

$$H = [a_1, b_1] \times \cdots \times [a_n, b_n]$$

obtained as a result of the branching process, we consider a population of agents (i.e., points within H). Given an agent $\mathbf{x} \in H$, a hyperrectangle

$$S^{\mathbf{x}} = S_1^{\mathbf{x}} \times \cdots S_n^{\mathbf{x}}$$

is associated to it, where each $S_i^{\mathbf{x}}$ is an interval centered at the corresponding component $\mathbf{x}[i]$ of the agent. The *size* of $S^{\mathbf{x}}$ is specified by the value $\rho(\mathbf{x})$: the i-th edge of $S^{\mathbf{x}}$ has length

$$2\rho(\mathbf{x}) \max\{b_i - \mathbf{x}[i], \mathbf{x}[i] - a_i).$$

As we will see, the ρ value associated to an agent is updated at each iteration according to some rule (see Sect. 4.6). The intersection $S^{\mathbf{x}} \cap H$ basically represents the local region around agent $\mathbf{x}$ which we want to explore. We also associate to each agent $\mathbf{x}$ an *effort* value $s(\mathbf{x})$, which specifies the amount of computational effort we want to dedicate to the exploration of $S^{\mathbf{x}}$ and is also updated at each iteration (see, again, Sect. 4.6).

Then, the subdomain H is *locally* explored by acquiring information about the landscape within each region $S^{\mathbf{x}}$ and *globally* explored by evolving a population of agents which are also allowed to collaborate with each other. Moreover, an archive X of solutions over the whole domain D is maintained during the search. The archive is maintained in order to have a set of low-lying local solutions for the problem at hand (see the discussion in the Introduction). The proposed approach, called Multiagent Collaborative Search, is outlined in what follows, while its details will be specified in the following subsections.

Multiagent collaborative search

Step 0. Initialization: Generate an initial population of agents P_0 within H through a Latin Hypercube (i.e., a non-collapsing design where points/agents are evenly spread even when projected along a single parameter axis; for a more detailed description and a justification of the use of Latin Hypercubes we refer, e.g., to [3]). A hyperrectangle $S^{\mathbf{x}_j^0}$ is associated to the j-th agent $\mathbf{x}_j^0 \in P_0$. The initial *size* $\rho(\mathbf{x}_j^0)$ of each region $S^{\mathbf{x}_j^0}$ is fixed to 1 (i.e., the initial local region of each agent corresponds to the whole set H). The *effort* $s(\mathbf{x}_j^0)$ dedicated to agent $\mathbf{x}_j^0 \in P_0$ is fixed to the same value $s_{\max}$ (equal to n in the computations) for all agents in P_0. Set $k = 0$.
Step 1. Collaboration: Agents collaborate with each other. The collaborations give rise to a set of new solutions denoted by Q_{k+1}. See Sect. 4.1.
Step 2. Selection: A subset of members of the set $P_k \cup Q_{k+1}$ will be selected to give rise to the new updated population P_{k+1}. See Sect. 4.2.
Step 3. Filtering: A *filter* partitions population P_{k+1} into two subsets P_{k+1}^{in}, the population within the filter, and P_{k+1}^{out}, the population outside the filter. See Sect. 4.3.
Step 4. Repulsion: The population is possibly further updated through a repulsion mechanism. See Sect. 4.4.
Step 5. Local actions: A set of actions, specified by a *behavior*, are applied to each agent $\mathbf{x} \in P_{k+1}$. These allow local exploration (within $S^{\mathbf{x}}$) of the region around the agent. They are repeatedly applied until either an improvement is observed or the number $s(\mathbf{x})$ of actions is reached. If an agent $\mathbf{x}$ generates an improvement, population P_{k+1} is updated by replacing $\mathbf{x}$ with its improvement. See Sect. 4.5.
Step 6. Hyperrectangle update: The size parameter ρ and the effort parameter s associated to each agent within the filter are updated according to some rule. See Sect. 4.6.
Step 7. Archive update: Apply filtering and update archive X (see Sect. 4.7).
Step 8. Stopping rule A stopping rule is checked (see Sect. 4.8). If it is not satisfied, then set $k = k + 1$ and go back to Step 1. If it is satisfied, then update the archive X by adding the current population, i.e., set $X = X \cup P_{k+1}$.

Note that in order to compare different solutions we need to define a selection criterion or a property qualifying each solution (fitness function). For the test problems considered in this paper, which can be reformulated as box-constrained ones, this property is simply the objective function. However, different definitions for problems with nonlinear constraints and multiple objectives have been proposed, e.g., in [5].

4.1 Collaboration

Collaboration defines operations through which information is exchanged between pairs of agents. Given a pair of agents $\mathbf{x}_1$ and $\mathbf{x}_2$, with $\mathbf{x}_1$ having better fitness value, three different operations are defined. Two of them are defined by adding to $\mathbf{x}_1$ a step $\mathbf{\Delta}$ defined as follows

$$\mathbf{\Delta} = \alpha_2 r^t (\mathbf{x}_2 - \mathbf{x}_1) + \alpha_1 (\mathbf{x}_2 - \mathbf{x}_1),$$

and correspond to: *extrapolation* on the side of $\mathbf{x}_1$ ($\alpha_1 = 0, \alpha_2 = -1, t = 1$), with the further constraint that the result must belong to the domain H (i.e., if the step $\mathbf{\Delta}$ leads out of H, its size is reduced until we get back to H); *interpolation* ($\alpha_1 = 0, \alpha_2 = 1$), where a random point between $\mathbf{x}_1$ and $\mathbf{x}_2$ is sampled. In the latter case, the shape parameter t is defined as follows:

$$t = 0.75 \frac{s(\mathbf{x}_1) - s(\mathbf{x}_2)}{s_{\max}} + 1.25$$

The rationale behind this definition is that we are favoring moves which are closer to the agent with higher fitness value if the two agents have the same s value, while in the case where the agent with highest fitness value has a s value much lower than that of the other agent, we try to move away from it because a small s value indicates that improvements close to the agent are difficult to detect.

The third operation is the *recombination* operator, a *single-point crossover*, where, given the two agents: we randomly select a component i; split the two agents into two parts, one from component 1 to component i and the other from component $i + 1$ to component n; and then we combine the two parts of each of the agents in order to generate two new solutions.

Note that three operations give rise to four new solutions denoted by $\mathbf{y}_1, \mathbf{y}_2, \mathbf{y}_3, \mathbf{y}_4$, called *children* ($\mathbf{x}_1$ and $\mathbf{x}_2$ are called *parents*).

The first of the two parents is selected at random in the worst half (from the point of view of the fitness) of the current population, while the second parent is selected at random in the whole population. We also tested the selection of the first parent from the best half but this reduced diversity of the population and caused premature convergence.

4.2 Selection

When collaboration holds between agents, we need to specify how the population evolves. As specified in Sect. 4.1, each pair of parents $\mathbf{x}_1$ and $\mathbf{x}_2$ generates four children $\mathbf{y}_1$, $\mathbf{y}_2$, $\mathbf{y}_3$ and $\mathbf{y}_4$. Then, a tournament, based on fitness values, is started between the worst of the two parents and the best of the four children. The winner of the tournament will be selected to enter the new population together with the best of the two parents. If the winner is one of the children, it will inherit the ρ and s values of the defeated parent.

4.3 Filtering

Given a population P_k a filter simply subdivides the population into two parts P_k^{in} and P_k^{out}. P_k^{in} contains the best members of the population P_k, i.e., those with the best fitness values, while P_k^{out} contains all the other individuals in P_k. The main difference between agents inside and outside the filter is that on agents outside the filter only mutation actions (see Eq. (12) below) over the whole subdomain H are performed (for each agent outside the filter the number of these mutation actions is a random one between 1 and the size of P_k^{out}), while also other actions, allowing a deeper local exploration, are performed on agents inside the filter (see the following Sect. 4.5). Moreover, values ρ and s are only updated for agents in P_k^{in} (see the following Sect. 4.6). In case an agent outside the filter at iteration k enters the filter at iteration $k+1$, its ρ and s values are initialized as specified in Step 0.

4.4 Repulsion

When the distance between two agents gets below a given threshold, a repulsion action is applied to the one with worse fitness value. More precisely, consider agent $\mathbf{x}_j$ and let

$$M_j = \{i \ : \ S^{\mathbf{x}_j} \cap S^{\mathbf{x}_i} \neq \emptyset, i \neq j\}$$

be the set of agents whose box has a nonempty intersection with the one of $\mathbf{x}_j$. Let $n_c(j)$ denote the cardinality of M_j. Then, for each $i \in M_j$ we check the following condition

$$w_c n_c(j)\rho(\mathbf{x}_j) > \rho_{ij},$$

where ρ_{ij} denotes the normalized distance[1] between $\mathbf{x}_i$ and $\mathbf{x}_j$ and w_c is a small positive parameter called *crowding factor*. If the condition is satisfied, then the worse between agents $\mathbf{x}_i$ and $\mathbf{x}_j$ is repelled (note that $w_c = 0$ corresponds to no repulsion). Repulsion is basically an interpolation between the agent to be repelled and one vertex of the current domain chosen at random. The idea behind repulsion is to avoid convergence of different agents to the same subregion with a consequent waste of computational effort.

4.5 Behavior

At every generation, a sequence of actions is performed by each agent, according to a behavior β. In particular, given agent j at generation k, denoted by $\mathbf{x}_j^k$, a behavior is a collection of displacement vectors $\Delta\xi$ generated by some function z_β:

$$\beta = \{\boldsymbol{\Delta}\xi \ \mid \mathbf{x}_j^k + \boldsymbol{\Delta}\xi \in H \quad \text{and} \quad \boldsymbol{\Delta}\xi = z_\beta(\mathbf{x}_j^k, \mathbf{x}_j^{k-1}, \mathbf{w}, \mathbf{r}, P_k)\} \tag{9}$$

where z_β is a function of the current and past state $\mathbf{x}_j^k$ and $\mathbf{x}_j^{k-1}$ of agent j, of a set of weights $\mathbf{w}$, of a set of random numbers $\mathbf{r}$ and of the current population P_k. Every point $\mathbf{x}_j^k + \boldsymbol{\Delta}\xi$ is called a *child* of agent j. In what follows we describe the different kinds of actions employed in this paper.

Inertia: This action is performed at most once at each generation. If agent j has improved from generation $k-1$ to generation k, then we follow the direction of the improvement (possibly until we reach the border of the hyperrectangle associated to the agent), i.e., we perform the following step:

$$\boldsymbol{\Delta}\xi = \bar{\lambda}(\mathbf{x}_j^k - \mathbf{x}_j^{k-1}) \tag{10}$$

where

$$\bar{\lambda} = \min\{1, \max\{\lambda \ : \ \mathbf{x}_j^k + \lambda(\mathbf{x}_j^k - \mathbf{x}_j^{k-1}) \in S^{\mathbf{x}_j}\}\}.$$

Follow-the-trail: This step is inspired by Differential Evolution (see, e.g., [17,21]). It is defined as follows: let $\mathbf{x}_{i_1}^k, \mathbf{x}_{i_2}^k, \mathbf{x}_{i_3}^k$ be three randomly selected agents; then

$$\boldsymbol{\Delta}\xi = \mathbf{x}_j^k - (\mathbf{x}_{i_1}^k + (\mathbf{x}_{i_3}^k - \mathbf{x}_{i_2}^k)) \tag{11}$$

(if the step leads out of $S^{\mathbf{x}_j}$, then its length is reduced until we reach the border of $S^{\mathbf{x}_j}$).

[1] By normalized distance we mean the distance between the two agents once H has been transformed into the unit hypercube through the appropriate affine transformation.

Random-Step (*or Mutation*): Given the agent $\mathbf{x}$ and its associated hyperrectangle $S^{\mathbf{x}}$, four different kinds of mutation actions are performed all arising from the following displacement of a component i of the agent:

$$\Delta\xi_i = w_1 r^t(\ell_i - x_i) + (1 - w_1) r^t(u_i - x_i) \tag{12}$$

where ℓ_i and u_i are respectively the lower and upper limits of x_i within $S^{\mathbf{x}} \cap H$, r is a uniform random number in [0, 1], $w_1 = 1$ with some probability p_i and $w_1 = 0$ with probability $1 - p_i$, and $t \geq 0$ is a shape parameter ($t = 1$ corresponds to uniform sampling, while $t > 1$ favors more local moves). The four mutation actions are the following:

- all components i are perturbed according to (12) with $t = 1$ and $p_i = 0.5$;
- a component i is selected at random and perturbed according to (12) with $t = 1$ and $p_i = 0.5$;
- a component i is selected at random and perturbed according to (12) with $t = 0.5$ and $p_i = 0.5$;
- a component i is selected at random and randomly fixed either at its lower limit or its upper limit in the region $S^{\mathbf{x}} \cap H$, i.e., $t = 0$ and $p_i = 0.5$.

Linear blending: Once a mutation action on agent $\mathbf{x}$ has been performed, its result, denoted by $\mathbf{y}$, is further refined through blending procedures. Linear blending corresponds to the following displacement:

$$\boldsymbol{\Delta}\xi = \alpha_2 r^t(\mathbf{y} - \mathbf{x}) + \alpha_1(\mathbf{y} - \mathbf{x}). \tag{13}$$

where $\alpha_1, \alpha_2 \in \{-1, 0, 1\}$, $r \in [0, 1]$ is a random number, and t a shaping parameter which controls the magnitude of the displacement. Here we use the parameter values $\alpha_1 = 0$, $\alpha_2 = -1$, $t = 1$, which corresponds to *extrapolation* on the side of $\mathbf{x}$, and $\alpha_1 = \alpha_2 = 1$, $t = 1$, which corresponds to *extrapolation* on the side of $\mathbf{y}$. If the displacement defined by an extrapolation action is too large, i.e., the resulting point is outside the hyperrectangle associated with the current agent, then it is reduced until the resulting point is within the hyperrectangle.

Quadratic blending: The outcome of the linear blending can be used to construct a second order local model of the fitness function. We can define a second order blending operator that generates a displacement using the agent $\mathbf{x}$, the perturbation $\mathbf{y}$ obtained by mutation, and the new point $\mathbf{z}$ generated by the linear blending operator. A second order one-dimensional model of the fitness function along the line with direction $\mathbf{x} - \mathbf{z}$ is obtained by fitting the fitness values in the three points $\mathbf{x}$, $\mathbf{y}$ and $\mathbf{z}$. Then, the new point is the minimum of the second-order model along the intersection of the line with the hyperrectangle associated with the agent.

As already pointed out, the inertia action is performed at most once. All the other actions are cyclically performed until either an improvement is observed or the number $s(\mathbf{x}_j^k)$ of actions is reached. Note that in each cycle only one of the four mutation actions is performed in turn.

4.6 Size and effort update

Given an agent $\mathbf{x}_j^k \in P_k^{in}$, its size parameter $\rho(\mathbf{x}_j^k)$, defining the hyperrectangle $S^{\mathbf{x}_j^k}$ centered at $\mathbf{x}_j^k$, and its effort parameter $s(\mathbf{x}_j^k)$, giving the maximum number of actions applied to it, are updated at each generation. Both are reduced or enlarged depending on whether an improvement has been observed or not in the previous generation.

If $\mathbf{x}_j^{k+1} \neq \mathbf{x}_j^k$, i.e., an improvement has been observed for agent j at iteration k, then the effort is updated according to the following formula:

$$s(\mathbf{x}_j^{k+1}) = \max\{s(\mathbf{x}_j^k) + 1, s_{\max}\},$$

i.e., it is increased by 1, unless the maximum allowed number of actions has been already reached (recall that in the computations $s_{\max}$ has been fixed to the dimension n of the problem). Basically, we are increasing the effort if the agent is able to improve. In the same case the size is increased by the following formula:

$$\rho(\mathbf{x}_j^{k+1}) = \max\{\rho(\mathbf{x}_j^k)\ln(e + rank(\mathbf{x}_j^{k+1})), 1\}$$

where $rank(\mathbf{x}_j^{k+1})$ is the ranking of the agent $\mathbf{x}_j^{k+1}$ within the population P_{k+1} (the best individual has rank equal to 1, the second best equal to 2, and so on). Basically, the worse the ranking of an individual, the greater the possible increase of the radius will be. The increase is limited from above by 1 (when $\rho = 1$ the local region around the agent to be explored is equal to the whole domain H). The idea is that for low ranked individuals it makes sense to look for larger improvements and then to try to find a better point in larger regions making the search more global.

If no improvement is observed, then the effort is updated according to the following formula:

$$s(\mathbf{x}_j^{k+1}) = \max\{s(\mathbf{x}_j^k) - 1, 1\},$$

i.e., it is decreased by 1, unless the minimum allowed number of actions has been already reached.

In the same case the size is reduced according to the following rule. Let $\rho_{min}(\mathbf{x}_j^k)$ be the smallest possible reduction of the size parameter such that the child $\mathbf{y}^*$ of $\mathbf{x}_j^k$ with best fitness value is still contained in the hyperrectangle. Then:

$$\rho(\mathbf{x}_j^{k+1}) = \begin{cases} \rho_{min}(\mathbf{x}_j^k) & \text{if } \rho_{min}(\mathbf{x}_j^k) \geq 0.5\rho(\mathbf{x}_j^k) \\ 0.5\rho(\mathbf{x}_j^k) & \text{otherwise} \end{cases}$$

i.e., the size parameter is reduced to $\rho_{min}(\mathbf{x}_j^k)$ unless this is smaller than $0.5\rho(\mathbf{x}_j^k)$, in which case we only halve the size parameter.

4.7 Archive update

Let ρ_{tol} be a small threshold value. Let $\mathbf{x}_i$ be an agent whose size parameter is below the threshold value, i.e., $\rho(\mathbf{x}_i) < \rho_{tol}$. Let L_i be the set of agents whose normalized distance from $\mathbf{x}_i$ is below the threshold ρ_{tol} (including $\mathbf{x}_i$ itself). If agent $\mathbf{x}_i$ is the best one (from the point of view of fitness) in L_i, then all agents in L_i are randomly regenerated within the current domain H and $\mathbf{x}_i$ is inserted in archive X, while if it is not, only agent $\mathbf{x}_i$ is randomly regenerated within H. At termination of the MACS algorithm we insert the whole final population into the archive.

4.8 Stopping rule

The stopping rule is, at the moment, quite simple: we stop the search within a subdomain when a prefixed number N_f of function evaluations is reached.

5 Computational results

The approach proposed in this paper was compared against a number of different approaches on two test cases: a minimum bi-impulsive transfer from the Earth to Apophis (see Sect. 2.1) and a minimum Δv transfer from the Earth to Saturn (see Sect. 2.2).

5.1 Earth-Apophis transfer

The launch date from the Earth has been taken in the interval [3653 10958] (number of elapsed days since January 1st, 2000), while the time of flight has been taken in the interval [50 900] days. A standard Lambert solver [1] coded in Matlab has been used to compute the transfer arc. Table 1 collects the result of all the tests performed.

A number of stochastic and deterministic based methods were applied to the search for the optimal launch date and transfer time. Each optimizer was run for an increasing number of function evaluations in the range from 1000 to 3000. The first and basic method is grid search: from the best point on the grid a local search was started. In Table 1 the grid search is called GRID and the values for each column correspond to the best solution found at the end of the local search. The naive grid search was then compared to two other well known deterministic solvers: DIRECT [16] (DIvided RECTangles) which implements a branch and prune technique and MCS [12] (Multilevel Coordinate Search) which implements a more sophisticated branch and prune algorithm with local search. DIRECT outperforms the simple grid search but yields the global optimum $f = 0.03669$ km/s only after 3000 evaluations. On the other hand MCS did not return any solution better than the GRID ones. This is mainly

Table 1 Comparison among several search algorithms on the Earth-Apophis transfer

Fun. Eval. Solver	1000	2000	3000
GRID (km/s)	1.2239	0.1617	0.1617
DIRECT (km/s)	0.1616	0.0618	0.0367
MCS (km/s)	0.1654	0.1648	0.1644
DEVEC (10 individuals)			
<0.037 (km/s)	3.00%(c)	36.00%(c)	51.00%(e)
<MCS	42.00%(c)	44.00%(c)	78.00%(e)
<DIRECT	38.00%(c)	40.00%(c)	41.00%(e)
GATBX			
<0.037 (km/s)	3.33%	5.0%	12.50%
<MCS	28.33%	27.5%	45.00%
<DIRECT	27.50%	18.33%	0.00%
BS			
<0.037 (km/s)	7.50%	9.17%	12.50%
<MCS	49.17%	58.33%	75.00%
<DIRECT	28.33%	30.83%	5.83%
MACS (10 Agents)			
<0.037 (km/s)	16.67%	62.50%	82.50%
<MCS	87.50%	90.00%	96.67%
<DIRECT	77.50%	82.50%	80.83%
PSO (10 particles)			
<0.037 (km/s)	1.00%	19.00%	36.00%
<MCS	35.00%	31.00%	44.00%
<DIRECT	32.00%	28.00%	34.00%
EPIC			
<0.037 (km/s)	–	–	98.33%

(c)—Convergence strategy for differential evolution; (e) Exploration strategy for differential evolution due to the fact that both DIRECT and MCS are sensitive to the initial sampling of the search space, which in turns depends on its boundaries.

Stochastic approaches were run 120 times for each number of function evaluations. In the design of a real mission, solutions with similar Δv values are practically equivalent. On the other hand it is very important to measure the reliability of a search method in returning a given solution. Therefore, we computed three quantities: the percentage of times the algorithm was returning a function value lower than 0.037 km/s, the percentage of times the algorithm was returning a function value lower than the solution provided by MCS for the same number of function evaluations and the percentage of times the algorithm was returning a function value lower than the solution provided by DIRECT for the same number of function evaluations.

Four stochastic methods were considered: a basic best start algorithm (called BS in the table) that samples uniformly the solution space and then starts a local search from the best sample, DEVEC [17] an implementation of differential evolution, GATBX a Matlab implementation of genetic algorithms [4] and PSO an implementation of Particle Swarm Optimization [7]. Since for each stochastic method a number of parameters needs to be set (for example number of generations and size of the population for genetic algorithms), different settings were tested and the best result found over all the tested settings has been reported. This is a very important point; in fact, it would be erroneous to use, for example, the same population size for two population-based methods like GA and DE since they work on different principles. From the results in Table 1 it can be seen how the stochastic methods provide on average a better answer than deterministic methods for a low number of function evaluations though they fail at guaranteeing convergence.

Moreover the simple best start approach is beaten by DEVEC on this test case but it is better than the genetic algorithm. The convergence and exploration strategies in DEVEC implement an action similar to (11), namely they generate a new individual as follows

$$\mathbf{x}_{i_1}^k + (\mathbf{x}_{i_3}^k - \mathbf{x}_{i_2}^k),$$

where $\mathbf{x}_{i_2}^k, \mathbf{x}_{i_3}^k$ are randomly selected individuals, while $\mathbf{x}_{i_1}^k$ is the best individual for the convergence strategy and a randomly selected one for the exploration strategy.

As it can be seen in Table 1, the former strategy converges too fast while the latter requires more function evaluations to converge but provides a better exploration of the solution space since it preserves diversity longer. MACS was applied using 10 agents with all of them performing the same actions, i.e., 10 *explorers* (the explorers are agents within the filter, all the other agents (if any) are outside the filter). The minimum size of the radius ρ was set to 1e-5 while the crowding factor ω_c was set equal to 1e-6. The table shows that the behavioral meta-heuristic proposed in this paper outperforms all the stochastic methods. Moreover, it outperforms also the deterministic methods reaching a very high probability of yielding the global optimum.

The behavioral approach therefore has improved, in this case, efficiency and reliability of the search compared to other methods. Finally the behavioral search was hybridized with the deterministic domain decomposition (EPIC, Evolutionary Programming and Interval Computation, in the table) and run for a total of 3000 function evaluations with σ equal to 0 (no run for a lower number of function evaluations since the hybridization is meant to improve reliability and not efficiency). In this case we counted the number of times the algorithm was able to identify the basin of attraction of the minimum. As can be seen the rate of success is almost 100. On the other hand if the parameter σ is put equal to 1 and the number of function

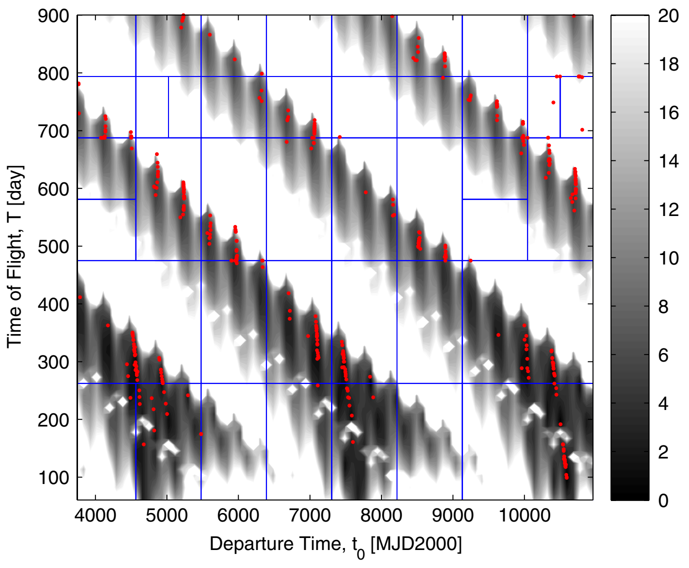

Fig. 1 Characterization of the solution space for the Earth-Apophis transfer. The plot represents the areas in the solution space with low Δv, dark grey, and high Δv, light grey. The Δv is in km/s. The dots are all the solutions in the archive X

evaluations are allowed to be up to 50000 then EPIC provides a characterization of most of the basins of attraction of the solution space (see Fig. 1).

5.2 MGA Earth-Saturn transfer

The second test is a multi gravity assist trajectory from the Earth to Saturn following the sequence Earth-Venus-Venus-Earth-Jupiter-Saturn. Gravity assist maneuvers have been modeled through a linked-conic approximation with powered maneuvers, i.e., the mismatch in the outgoing velocity is compensated through a Δv maneuver at the GA planet. No deep-space maneuvers are possible and each planet-to-planet transfer is computed as the solution of a Lambert's problem. The objective function is given in (4). An optimal solution for this problem was efficiently obtained through the combination of a space pruning technique combined with differential evolution [15]. In the same work it was proven that a deterministic algorithm exists that allows an efficient solution in polynomial time with a low exponent. However the algorithm is problem dependent and cannot be applied to a general black-box problem.

Here we apply all the above mentioned algorithms for an increasing number of function evaluations in the range from 20000 to 320000. Stochastic algorithms have been run for 500 independent times.

The best-known solution, found with the MACS algorithm and published on the ESA/ACT web site (the black-box objective function can be downloaded from the ESA/ACT web site http://www.esa.int/gsp/ACT/mission_analysis/GlobalOptimisationProblems/EVVEJS.htm), is 4.9307 km/s. A number of local minima exists with a value slightly higher than 5 km/s. Therefore for stochastic methods we computed the percentage of times the algorithm was returning a function value lower than 5 km/s. As for the previous case, different settings have been tested for each method and each number of function evaluations.

Table 2 Comparison among several search algorithms on the MGA case

Fun. Eval./Solver	20000	40000	80000	160000	320000
DIRECT (km/s)	8.168	8.155	8.08	7.774	7.726
MCS (km/s)	26.64	9.042	9.042	7.377	7.376
DEVEC					
<5	1.00%	0.50%	1.00%	1.25%	1.00%
<MCS	100.0%	28.75%	26.75%	25.25%	26.75%
<DIRECT	28.00%	27.75%	26.00%	25.25%	26.75%
GATBX					
<5	0.00%	0.00%	0.00%	0.25%	0.00%
<MCS	100.0%	69.70%	73.25%	65.25%	64.00%
<DIRECT	59.50%	65.70%	70.00%	68.50%	65.50%
BS					
<5	0.00%	0.25%	0.50%	0.50%	0.25%
<MCS	100.0%	93.50%	99.75%	99.75%	99.75%
<DIRECT	70.25%	90.00%	98.50%	99.75%	100.0%
PSO					
<5	0.00%	0.00%	0.60%	0.40%	0.40%
<MCS	84.50%	42.50%	23.20%	19.00%	23.40%
<DIRECT	33.25%	41.50%	21.80%	19.00%	23.40%
MACS					
<5	2.25%	7.75%	6.89%	6.00%	10.25%
<MCS	100.0%	64.50%	62.00%	93.00%	99.75%
<DIRECT	47.25%	61.00%	59.25%	93.00%	99.75%
EPIC					
<5	–	–	–	–	18.40%
<MCS	–	–	–	–	76.00%
<DIRECT	–	–	–	–	76.00%

In particular for DEVEC, PSO and MACS we ran the same case with a population of 20, 40 and 80 individuals. GATBX implements a multipopulation strategy for GA, therefore the total number of individuals was increased from 120 to 480 with different numbers of subpopulations: 2, 4 and 8. Table 2 reports the best results obtained for each number of function evaluations and each algorithm. No grid search is reported since no good results were obtained even with a fine initial grid made of 10^6 points.

Up to 160000 function evaluations, DEVEC, PSO and MACS yielded the best results with a population of size 20, after that the population of was extended to 40 (but with 20 explorers for MACS). An increase of the population size for DEVEC with exploration strategy yielded a significant increase in the percentage of times DEVEC is better than MCS and DIRECT, up to 87%, but reduced the convergence to values lower than 5 km/s and therefore it was not reported in Table 2. The one reported in the table is instead the convergence strategy since it converged to values lower than 5 km/s. Note that the exploration strategy even for a number of function evaluations lower than 160000 gave remarkably better results than MCS and DIRECT in 65% of the runs, very similar to GATBX.

Furthermore, both DIRECT and MCS cannot find a solution even close to the best solution found so far. We also pushed DIRECT up to 8 million function evaluations obtaining only a value of 5.5879 km/s. On the other hand when MACS is allowed to perform 320000 function evaluations, 99.75% of the times it converges to a value which is lower than 5.32 km/s, which is close to a very strong attractor, while all the other optimizers remain, on average, quite above this value. Furthermore an increase in the number of function evaluations is not beneficial for almost each one of the stochastic methods except for MACS that maintains an adequate level of diversity. It is also remarkable that a simple best start approach is as effective as DE and how both DE and PSO yield similar results, in both cases better than GA.

Finally we applied the domain decomposition technique to the MGA case (EPIC in the table). A population of 20 agents of which 20 explorers was deployed in the search space. The maximum total number of function evaluations was fixed to 320000 which corresponds to a maximum of 40000 function evaluations for every subdomain and a total of $\tau = 8$ subdomains. The parameter σ was set at first equal to 1 to have full exploration. The result was a slight increase, up to 12%, in the percentage of times the algorithm was able to find the global minimum. A second test was performed with 40 agents of which 20 explorers. The parameter σ was set equal to 0 to have full convergence and the result is reported in Table 2. For all the MGA tests with both MACS and EPIC, the minimum size of the radius was set to 1e-5 while the crowding factor ω_c was set equal to 1e-4.

6 Conclusions

In this paper we proposed a global optimization approach for solving optimal trajectory design problems. The approach is based on domain decomposition and each subdomain is evaluated through a stochastic multiagent approach based on behaviorism and on action-selection. The approach has been tested and compared with other stochastic and deterministic methods on two global trajectory problems. On the simpler, two-dimensional, problem it has demonstrated a higher robustness compared to all the other stochastic methods, reaching more than 80% success rate. On the more complex problem it is one order of magnitude more reliable than the tested stochastic methods and outperforms the deterministic ones.

References

1. Battin, R.H.: An Introduction to the Mathematics and Methods of Astrodynamics, revised edition (Aiaa Education Series). AIAA (American Institute of Aeronautics & Ast) revised edition (1999). ISBN-13: 978-1563473425
2. Dachwald, B.: Optimization of interplanetary solar sailcraft trajectories using evolutionary neurocontrol. J. Guid. Dyn. January/ February (2004)
3. Dam, E.R., van Husslage, B.G.M., den Hertog, D., Melissen, J.B.M.: Maximin Latin hypercube designs in two dimensions. Oper. Res. **55**, 158–169 (2007)
4. Chipperfield, A.J., Fleming, P.J., Pohlheim, H., Fonseca, C.M.: Genetic Algorithm Toolbox User's Guide, ACSE Research Report No. 512, University of Sheffield (1994)
5. De Pascale, P., Vasile, M.: Preliminary design of low-thrust multiple gravity assist trajectories. J. Spacecr. Rockets **43**(5), 1065–1076 (2006)
6. Di Lizia, P., Radice, G.: Advanced global optimisation tools for mission analysis and design. European Space Agency, the Advanced Concepts Team, Ariadna Final Report 03-4101b (2004)
7. Elbeltagi, E., Hegazy, T., Grierson, D.: Comparison among five evolutionary-based optimization algorithms. Adv. Eng. Informatics **19**, 43–53 (2005)
8. Gage, P.J., Braun, R.D., Kroo, I.M.: Interplanetary trajectory optimisation using a genetic algorithm. J. Astronaut. Sci. **43**(1), 59–75 (1995)
9. Gurfil, P., Kasdin, N.J.: Niching genetic algorithms-based characterization of geocentric orbits in the 3D elliptic restricted three-body problem. Comput. Methods Appl. Mech. Eng. **191**(49–50), 5673–5696 (2002)
10. Hartmann, J.W., Coverstone-Carrol, V.L., Williams, S.N.: Optimal interplanetary spacecraft trajectories via pareto genetic algorithm. J. Astronaut. Sci. **46**(3), 267–282 (1998)
11. Horst, R., Tuy, H.: Global Optimization: Deterministic Approaches, 3rd edn. Springer Verlag, Berlin (1996)
12. Huyer, W., Neumaier, A.: Global optimization by multilevel coordinate search. J. Glob. Optim. **14**, 331–355 (1999)

13. Jones, D.R.: A taxonomy of global optimization methods based on response surfaces. J. Glob. Optim. **21**, 345–383 (2001)
14. Labunsky, A.V., Papkov, O.V., Sukhanov, K.G.: Multiple Gravity Assist Interplanetary Trajectories. ESI Book Series (1998)
15. Myatt, D.R., Becerra, V.M., Nasuto, S.J., Bishop, J.M.: Advanced Global Optimization Tools for Mission Analysis and Design. Final Rept. ESA Ariadna ITT AO4532/18138/04/NL/MV,Call03/4101 (2004)
16. Perttunen, C.D., Stuckman, B.E.: Lipschitzian optimization without the Lipschitz constant. JOTA **79**(1), 157–181 (1993)
17. Price, K.V., Storn, R.M., Lampinen, J.A.: Differential Evolution: A Practical Approach to Global Optimization, 1st edn. Springer (2005). ISBN-13: 978-3540209508
18. Rauwolf, G., Coverstone-Carroll, V.: Near-optimal low-thrust orbit transfers generated by a genetic algorithm. J. Spacecr. Rockets **33**(6), 859–862 (1996)
19. Rogata, P., Di Sotto, E., Graziano, M., Graziani, F.: Guess Value for Interplanetary Transfer Design Through Genetic Algorithms. American Astronautical Society, AAS Paper 03-140 (2003)
20. Stephens, C.P., Baritompa, W.: Global optimization requires global information. J. Optm. Theory Appl. **96**, 575–588 (1998)
21. Storn, R., Price, K.: Differential evolution—a simple and efficient heuristic for global optimization over continuous spaces. J. Glob. Optim. **11**(4), 341–359 (1997)
22. Torn, A., Zilinskas, A.: Global Optimization. Springer, Berlin (1987)
23. Vasile, M.: Combining evolution programs and gradient methods for WSB transfer optimisation. In: Operational Research in Space & Air, vol. 79. Book Series in Applied Optimization Kluwer Academy Press (2003). ISBN 1-4020-1218-7